МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический университет
Петра Великого»

Институт компьютерных наук и технологий

Высшая школа технологий искусственного интеллекта

Направление: 02.03.01 Математика и компьютерные науки

Отчет о выполнении курсовой работы «Реализация приложения телефонный справочник на языке C++ с использованием фреймворка Qt» Курс «Объектно-ориентированное программирование»

Выполнил студент группы

№5130201/30001 _____ Мелещенко С. И.

Преподаватель _____ Глазунов В. В.

«_____»_____ 2024г.

Санкт-Петербург, 2024

Содержание

Введение

Приложение "Телефонный справочник"представляет собой интерактивную таблицу, предназначенную для хранения и управления контактными данными. В таблице могут содержаться следующие сведения: имя, фамилия, отчество, адрес, дата рождения, email и телефонные номера (рабочий, домашний, служебный) в любом количестве. Основные функции приложения включают:

1) добавление новых контактов;
2) редактирование и удаление существующих записей;
3) сортировку и поиск по любому полю;
4) сохранение данных в файл и загрузку данных из файла.

Телефонные справочники представляют собой класс приложений, обеспечивающих удобное управление контактной информацией. Такие приложения актуальны для личного использования, ведения базы клиентов в малом бизнесе или в качестве прототипа для более сложных систем CRM (систем управления взаимоотношениями с клиентами).

Для реализации данного приложения был выбран фреймворк Qt, который предоставляет обширные возможности для разработки графических интерфейсов, работы с данными и обеспечения кроссплатформенной совместимости.

Qt — это библиотека классов C++ и набор инструментального программного обеспечения для создания кросс-платформенных приложений с графическим интерфейсом (GUI). Существуют вариации для других языков: PyQt для Python, QtRuby для Ruby, Qt Jambi для Java.

Преимущества и возможности фреймворка Qt.

1. Интуитивный интерфейс пользователя. Qt предлагает создание виджетов и инструментов для создания удобного интерфейса, позволяет разработчику реализовать такие функции, как добавление, редактирование, сортировка и поиск контактов, обеспечивая простоту взаимодействия для пользователя.

2. Работа с данными. Qt предоставляет классы для работы с данными. Например, QAbstractTableModel облегчает отображение списков контактов и работу с ними.

3. Валидация данных. Использование QRegularExpression упрощает проверку корректности вводимой информации.

4. Кроссплатформенность. Qt позволяет создавать приложения, которые одинаково работают на Windows, macOS и Linux.

5. Поддержка событийной модели. Qt реализует событийно-ориентированную архитектуру, что позволяет создание динамических и интерактивных приложений.

6. Гибкость и масштабируемость. Приложения, созданные на Qt, легко адаптируются к изменениям требований.

Использование фреймворка Qt для создания "Телефонного справочника"позволило успешно реализовать все требуемые функции, а также обеспечило удобство разработки и возможность дальнейшего расширения функционала.

1 Постановка задач

Цель: Создать приложение "Телефонный справочник"с возможностью хранения и работы с контактами с помощью Qt на языке C++.

Задачи.

1. Реализовать хранение информации о контактах с обязательными полями:
   a) имя (тип QString);
   b) фамилия (тип QString);
   c) отчество (тип QString);
   d) адрес (тип QString);
   e) дата рождения (тип QDate);
   f) электронная почта (тип QString);
   g) телефонные номера (тип QStringList).

2. Реализовать проверку корректности вводимых данных с помощью регулярных выражений. Проверка осуществляется для следующих полей.

   a. Имя. Содержит только буквы, цифры, дефис или пробел, начинается с большой буквы и заканчивается буквой или цифрой.

   b. Фамилия. Содержит только буквы, цифры, дефис или пробел, начинается с большой буквы и заканчивается буквой или цифрой.

   c. Отчество. Содержит только буквы, цифры, дефис или пробел, начинается с большой буквы и заканчивается буквой или цифрой.

   d. Телефонные номера. Начинаются с + или цифры, содержат от 3 до 11 символов и все символы после первого — только цифры.

   e. Электронная почта. Содержит имя пользователя, символ @ и доменное имя. Имя пользователя: латинские буквы, цифры, точка или подчеркивание. Доменное имя: латинские буквы, цифры, точки или подчеркивание.

   f. Дата рождения. Меньше текущей даты или равна ей.

3. Реализовать возможность добавления контактов с помощью кнопки "Add Contact". При нажатии на кнопку открывается диалоговое окно с полями ввода. Введенные данные проверяются на корректность, и только после успешной валидации контакт добавляется в таблицу.

4. Реализовать возможность изменения контактов с помощью выбора одной записи в таблице и вызова контекстного меню правой кнопкой мыши. В меню доступна опция "Edit Contact". Открывается диалоговое окно с предзаполненными данными выбранного контакта. После редактирования пользователь нажимает "OK и только после успешной валидации изменения сохраняются.

5. Реализовать возможность удаления контактов. Пользователь может выбрать один или несколько контактов в таблице и вызвать контекстное меню нажатием правой кнопки мыши. Опция "Delete Contacts"вызывает подтверждение действия через диалоговое окно. После подтверждения выбранные контакты удаляются.

6. Реализовать поиск контактов по любому полю. Поиск осуществляется по подстроке или строке в любом поле контакта. Введенный текст сравнивается с данными в таблице без учета регистра. Результаты поиска отображаются путем скрытия строк, которые не содержат совпадений.

7. Реализовать сортировку контактов по любому полю. Сортировка выполняется по возрастанию или убыванию по любому из полей по выбору пользователя при нажатии на заголовок нужного столбца. Одно нажатие соответствует сортировке по возрастанию, второе - по убыванию.

8. Обеспечить сохранение данных за счет записи имеющейся информации в файл. Контакты сохраняются в текстовом файле с разделителем запятая между полями. Телефон-

ные номера внутри одной записи разделяются точкой с запятой. Формат даты рождения — YYYY-MM-DD.

9. Обеспечить возможность считывания данных из файла в приложение. Программа считывает контакты из текстового файла формата *.txt. Если данные в файле некорректны, они также некорректно отображаются в приложении.

Ограничения.

1. Имя, Фамилия, Отчество.
   a. Должны содержать только буквы, цифры различных алфавитов, дефис и пробел.
   b. Начинаются с большой буквы.
   c. Заканчиваются только на букву или цифру.
2. Номер телефона.
   a. Начинается только с + или цифры.
   b. Содержит от 3-х (для экстренных номеров) до 11-и (номера, начинающиеся на +7) символов.
   c. Плюс может быть только первым знаком, остальная часть это обязательно цифры.
3. Дата рождения меньше текущей даты или равна ей.
4. Элетронная почта состоит из имени пользователя, символа "@"и имени домена.
   a. Имя пользователя может состоять только из латинских букв, цифр, точки, символа нижнего подчеркивания.
   b. Обязательно наличие одного символа "@".
   c. Доменная часть состоит только из латинских букв, цифр, точки и подчёркивания.
5. При вводе некорректных данных запись не сохраняется.
6. Нельзя выбрать сразу несколько полей и нажать на кнопку "изменить".
7. При поиске несуществующей записи, ничего не отображается.
8. В файле, из которого считываются контакты поля должны быть разделены запятой (должны отсутствовать пробелы). Если номеров телефона несколько они разделяются точкой с запятой (должны отсутствовать пробелы). При некорректной записи в приложении будут также некорректные данные.
9. Нельзя ничего изменить или удалить, пока не выбран контакт.
10. Запись и считывание осуществляется с помощью файла *.txt.
11. Дата рождения в файле хранится в формате YYYY-MM-DD.

2 Реализация

Программа помимо main состоит из трех основных классов.

1. class MainWindow : public QMainWindow — основной класс пользовательского интерфейса, наследуемый от QMainWindow.

2. class PhoneBookModel : public QAbstractTableModel — класс, наследуемый от QAbstractTableModel.

3. class Contact — структура для хранения информации о контакте.

А также из функций, не привязанных к классам, которые с помощью регулярнных выражений проверяют корректность ввода:

1) bool validateName(const QString& name);
2) bool validateEmail(const QString& email);
3) bool validatePhoneNumber(const QString& phone);
4) bool validateBirthDate(const QDate& date);
5) bool validateContact(const Contact& contact)

Рассмотрим каждую часть подробнее.

2.1 Библиотечные классы

QFileDialog - предоставляет стандартный диалог для открытия и сохранения файлов. Позволяет пользователю выбирать файлы или папки.

QMessageBox - используется для отображения сообщений, предупреждений, ошибок или запроса подтверждения от пользователя.

QInputDialog- позволяет пользователю вводить строковые, числовые или другие значения через простой диалог.

QVBoxLayout / QHBoxLayout - организуют виджеты вертикально (QVBoxLayout) или горизонтально (QHBoxLayout) в контейнере.

QDateEdit - виджет для ввода и редактирования дат, поддерживает выбор даты через выпадающий календарь.

QApplication - основной класс, управляющий жизненным циклом приложения и его событиями.

QDialog - базовый класс для создания диалоговых окон, которые могут быть модальными или немодальными.

QFormLayout - предназначен для упорядочивания виджетов в форме "ярлык — поле ввода".

QDialogButtonBox - управляет стандартными кнопками (например, "OK "Cancel") в диалоговых окнах.

QDate - класс для работы с датами, поддерживает форматирование, вычисления и проверку корректности.

QString - представляет строки текста с поддержкой Unicode и методов для работы с текстовыми данными.

QRegularExpression - класс для работы с регулярными выражениями, используется для валидации и поиска текста.

QMainWindow - основной класс для создания окон приложения с поддержкой меню, инструментальных панелей и центрального виджета.

QMenu - создает выпадающее меню для приложений, поддерживает действия (QAction) и вложенные меню.

QAbstractTableModel - базовый класс для моделей данных, представленных в табличной форме. Предназначен для работы с QTableView.

QFile - управляет работой с файлами, предоставляет интерфейс для чтения и записи данных.

QTextStream - упрощает чтение и запись текстовых данных в файлы или строки.

QPushButton - виджет кнопки, который может использоваться для выполнения действий или отправки сигналов.

QLineEdit - виджет однострочного текстового ввода, позволяет вводить или редактировать текст.

QTableView - виджет для отображения данных в табличной форме с поддержкой сортировки, выбора и прокрутки.

QLabel - используется для отображения текста или изображений.

2.2 Main

Функция int main(int argc, char* argv[]) запускает наше приложение.

QApplication app(argc, argv) — создается объект QApplication, он обрабатывает параметры командной строки и управляет основным циклом событий.

MainWindow window; — создается объект главного окна приложения, который является экземпляром класса MainWindow.

window.show() — отображает главное окно на экране.

return app.exec() — запускает основной цикл обработки событий, который продолжает работать до тех пор, пока приложение не будет закрыто. Этот цикл управляет всеми взаимодействиями с интерфейсом пользователя.

2.3 Класс MainWindow

Класс MainWindow наследуется от QMainWindow — базового класса в Qt для создания главного окна приложения. Это необходимо для использования встроенных возможностей: работа с меню, панели инструментов, centralWidget и системой компоновки (layout), управление событиями, происходящими в окне, через сигнально-слотовую систему.

2.3.1 Поля класса

PhoneBookModel* phoneBookModel - для управления данными телефонного справочника.

QTableView* tableView - для отображения данных справочника в виде таблицы.

QMenu* contextMenu - для доступа к действиям "Редактировать"и "Удалить"при правом клике на таблицу.

QLineEdit* searchLineEdit - ввод строки поиска.

Кнопки управления.
QPushButton* addButton — добавление контакта.
QPushButton* loadButton — загрузка данных из файла.
QPushButton* saveButton — сохранение данных в файл.
QPushButton* searchButton — запуск поиска.
QPushButton* clearSearchButton — очистка результатов поиска.

2.3.2 Конструктор

MainWindow::MainWindow(QWidget* parent) : QMainWindow(parent),
phoneBookModel(new PhoneBookModel(this))
parent — указатель на родительский виджет.
phoneBookModel — создается модель для управления контактами.

Вызов вспомогательных методов setupUI для настройки пользовательского интерфейса, setupConnections для установки соединений сигналов и слотов. Также установка стиля приложения.

2.3.3 Метод setupUI

void MainWindow::setupUI() создает и настраивает пользовательский интерфейс.

Создается объект centralWidget, который будет базовым контейнером для всех элементов интерфейса. Метод setCentralWidget устанавливает его как главный виджет в окне MainWindow.

Создаются кнопки как объекты класса QPushButton, они реализуют базовые операции приложения (добавление, загрузку, сохранение, поиск контактов и очистку поиска).

Создается объект QLineEdit, который представляет собой строку ввода. Метод setPlaceholderText добавляет текст, который отображается в пустой строке ввода.

Создается таблица QTableView. setModel(phoneBookModel) связывает таблицу с моделью данных PhoneBookModel (определяет, как данные будут представлены в таблице).

setSortingEnabled(true) включает возможность сортировки по столбцам.

setContextMenuPolicy(Qt::CustomContextMenu) - контекстное меню для таблицы.

Вызывается функция setupContextMenu, которая добавляет функционал контекстного меню (позволяет выполнять операции над выбранными строками таблицы).

Создается объект QLabel с текстом "Telephone directory". Жирный шрифт размера 20 пунктов. Устанавливается цвет текста через setStyleSheet.

Создается вертикальный макет (QVBoxLayout), в который по порядку добавляются заголовок, макет строки поиска и таблица. Таким образом объекты будут выстроены друг под другом.

Создается горизонтальный макет (QHBoxLayout), в который добавляются три кнопки: добавление, загрузка и сохранение. Таким образом, они выстраиваются в один ряд.

Горизонтальный макет кнопок добавляется в вертикальный основной макет, готовый макет закрепляется за центральным виджетом.

2.3.4 Метод setupConnections

void MainWindow::setupConnections() связывает сигналы (события) интерфейсных элементов с соответствующими слотами (методами), реализующими функциональность приложения.

2.3.5 Метод searchLayout

QHBoxLayout* MainWindow::searchLayout() создает горизонтальный компоновочный макет (QHBoxLayout) для строки поиска и кнопок, связанных с поиском.

2.3.6 Метод getContactFromUser

Contact MainWindow::getContactFromUser(const QString& title, const Contact& defaultContact) выводит диалоговое окно для ввода или изменения данных контакта, возвращает объект Contact с заполненной информацией.

Создается модальное окно QDialog, которое блокирует взаимодействие с основным окном до завершения работы. Заголовок окна задается переданным параметром title.

Создается QFormLayout, компоновка, которая организует элементы ввода в виде полей с метками.

Создаются поля ввода данных контакта для имени, фамилии, отчества, адреса и email.

Создается поле для ввода даты рождения.

setCalendarPopup(true) - активирует календарь.

setDisplayFormat("dd.MM.yyyy") - устанавливается формат отображаемой даты.

Создается вертикальный компоновочный макет для размещения полей ввода телефонов. Затем если контакт уже содержит телефоны, для каждого номера создается QLineEdit, если у контакта нет номеров телефонов, добавляется одно пустое поле для ввода.

После этого создается кнопка для добавления дополнительных номеров телефонов, при нажатии на кнопку добавляется новое поле для ввода телефона и добавляются строки с полями ввода в форму. В конце номера телефонов добавляются в отдельный виджет, чтобы оформить их в виде вертикального списка.

Создается кнопка OK и Cancel для подтверждения или отмены ввода данных.

Выполняется валидация всех полей. Если все данные валидны, окно закрывается. Если есть ошибки, выводится сообщение с перечнем проблем.

При нажатии кнопки Cancel закрывает окно без сохранения изменений.

Если данные валидны и окно закрыто с помощью OK, создается новый контакт. Создается и возвращается объект Contact, заполненный данными, введенными пользователем.

Если диалог был отклонен, возвращается исходный контакт.

2.3.7 Метод addContact

void MainWindow::addContact() выполняет добавление нового контакта в телефонную книгу.Отображается диалоговое окно для ввода данных о новом контакте, затем объект contact добавляется в модель phoneBookModel с помощью метода addContact.

2.3.8 Метод loadContacts

Метод void MainWindow::loadContacts(). Первая строка вызывает диалоговое окно QFileDialog::getOpenFileName, которое позволяет пользователю выбрать файл для загрузки. Если путь к файлу не пустой, то вызывается метод loadFromFile у модели phoneBookModel, которая занимается загрузкой данных из выбранного файла.

2.3.9 Метод saveContacts

Метод void MainWindow::saveContacts() сохраняет текущие данные о контактах в файл. Сначала вызывается диалоговое окно QFileDialog::getSaveFileName, которое позволяет пользователю выбрать местоположение и ввести имя файла для сохранения данных. Если путь к файлу не пустой, вызывается метод saveToFile у модели phoneBookModel для сохранения текущих данных в указанный файл.

2.3.10 Метод searchContacts

void MainWindow::searchContacts() выполняет поиск по всем данным в модели телефонной книги, чтобы отфильтровать строки, которые содержат указанный пользователем поисковый запрос.

В первой строке извлекается текст поискового запроса из поля QLineEdit. Далее начинается цикл, который проходит по всем строкам в phoneBookModel. Внутри цикла по строкам идет еще один цикл, который перебирает все столбцы в строке. Для каждой ячейки создается QModelIndex, который указывает на конкретную ячейку в модели (по строке и столбцу). Метод data(index) возвращает данные в ячейке. Метод contains() проверяет, содержит ли эта строка поисковый запрос searchTerm, игнорируя регистр (Qt::CaseInsensitive). Если найдено совпадение с поисковым запросом, устанавливается флаг match = true, и

поиск по остальным столбцам этой строки прекращается. Если match ==true, то эта строка остается видимой в таблице. В противном случае строка скрывается с помощью метода setRowHidden(row, true).

Ниже приведена блок-схема метода (см. [Рис. 1]).

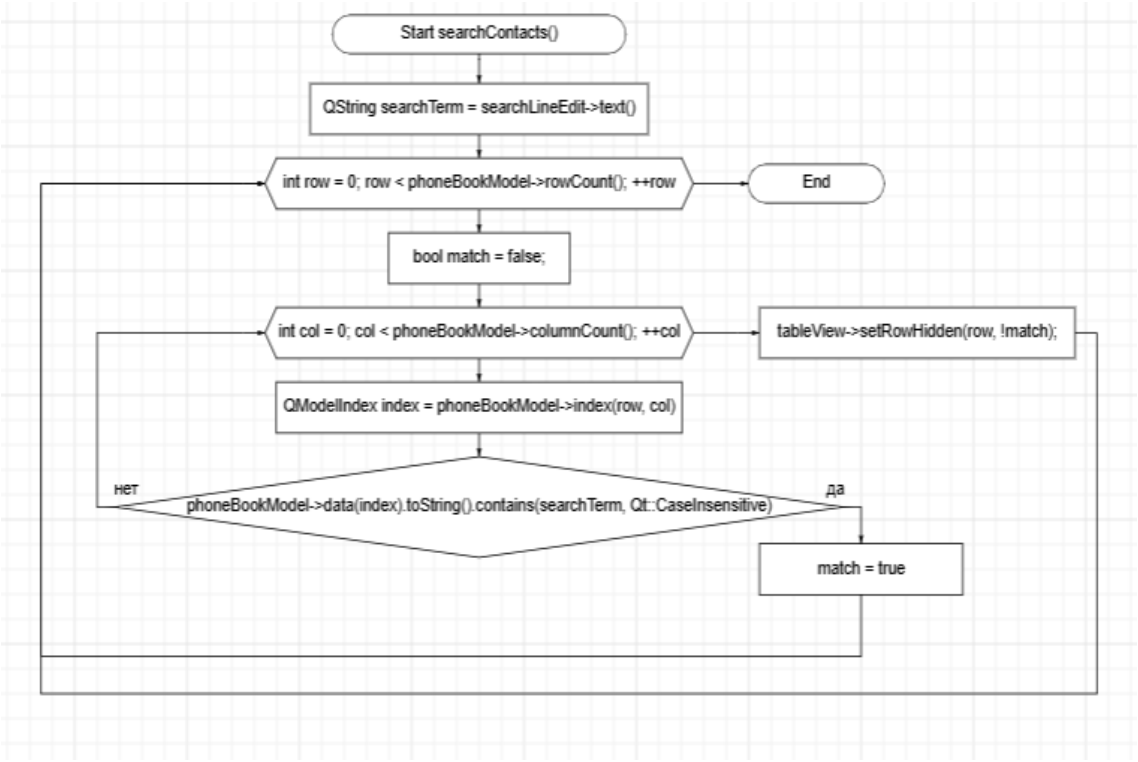

Рис. 1: Блок-схема поиска

2.3.11 Метод clearSearch

Метод void MainWindow::clearSearch() выполняет операцию сброса поиска и восстанавливает отображение всех строк в таблице. Сначала метод очищает поле ввода, затем цикл проходит по всем строкам и цикл, который проходит по всем строкам, и setRowHidden(row, false) устанавливает видимость строки в таблице

2.3.12 Метод setupContextMenu

void MainWindow::setupContextMenu() настраивает контекстное меню для таблицы с контактами.

Создается объект QMenu, который будет отображать контекстное меню, создаются два действия: редактирование контакта и удаление контактов, сигналы от действий связываются с методами.

Настроено отображение контекстного меню при правом клике на строку в таблице. Если клик происходит на действительной строке, меню отображается в глобальных координатах.

2.3.13 Метод editSelectedContact

Метод void MainWindow::editSelectedContact() отвечает за редактирование выбранного контакта.

Получаем выбранные строки и проверяем их количество. Если строка одна, то выполняется редактирование. В противном случае выводится предупреждение.

Редактирование одной строки: определяем индекс выделенной строки и получаем соответствующий контакт из модели, открывается диалоговое окно для редактирования контакта, текущие данные передаются как начальные значения. Затем удаляем старый контакт и добавляем обновленный.

2.3.14 Метод deleteSelectedContacts

Метод void MainWindow::deleteSelectedContacts() реализует удаление выбранных контактов из телефонной книги.

Для начала получаем выбранные строки, проверяем на пустой выбор. Если выбор пустой - предупреждение.

У пользователя запрашивается подтверждение удаления. Если он выбирает No, метод завершает работу.

Индексы сортируются в порядке убывания, чтобы удаление контактов начиналось с конца. Это предотвращает смещение индексов и ошибки при удалении. Каждый выделенный контакт удаляется из модели данных.

2.4 Класс PhoneBookModel

Класс PhoneBookModel предоставляет реализацию модели данных для телефонной книги. Он наследуется от QAbstractTableModel, что позволяет отображать и управлять данными в виде таблицы. Основная задача этого класса — управление списком контактов, их отображение, сохранение, загрузка и сортировка.

2.4.1 Поля класса

QList<Contact> contacts - список объектов Contact, представляющих данные телефонной книги.

2.4.2 Конструктор

Ronструктор класса PhoneBookModel инициализирует объект модели, передавая родительский объект в конструктор базового класса QAbstractTableModel.

2.4.3 Метод saveToFile

void PhoneBookModel::saveToFile(const QString& filename) сохраняет список контактов в *.txt файл в текстовом формате. Каждый контакт записывается в строку, где поля разделены запятыми, а телефонные номера внутри одной строки — точкой с запятой. После каждого контакта добавляется символ новой строки.

Создаётся объект класса QFile, который связывается с именем файла, переданным в параметре filename. Файл открывается в режиме записи с использованием флага

QIODevice::WriteOnly. Если файл не удаётся открыть, метод завершается без выполнения дальнейших действий. Объект QTextStream связывается с файлом для упрощённой записи текста в файл, циклом проходим по contacts, поля записываются в файл. После каждого контакта добавляется символ новой строки. После записи данных файл закрывается.

2.4.4 Метод loadFromFile

Метод loadFromFile(const QString& filename) выполняет загрузку списка контактов из текстового файла *.txt. Контакты считываются построчно, каждая строка файла интерпретируется как информация о единственном контакте. Поля разделены запятыми, а номера телефонов точкой с запятой.

Создаётся объект QFile, привязанный к пути, переданному в filename, если открыть файл не удалось, метод завершает выполнение.

Текстовый поток QTextStream используется для последовательного чтения данных из файла. Инициализируется временный список для хранения прочитанных контактов, далее следует чтение: метод readLine() считывает одну строку из файла, строка разделяется на части (поля) с помощью split(,).

Если количество элементов в data меньше 7, строка пропускается. Поля из строки используются для создания объекта Contact. Дата рождения преобразуется из строки в объект QDate в формате ISODate. Телефонные номера разбиваются по символу точка с запятой. Затем файл закрывается. а модель обновляется.

2.4.5 Метод addContact

Метод void PhoneBookModel::addContact(const Contact& contact) добавляет новый контакт в список контактов модели, с предварительной валидацией всех данных. Если данные контакта некорректны, добавление не производится.

2.4.6 Метод removeContact

Метод void PhoneBookModel::removeContact(int row). После проверки корректности индекса метод удаляет контакт из списка на указанной строке (индексе), одновременно уведомляя представление о внесённых изменениях.

2.4.7 Метод rowCount

Метод int PhoneBookModel::rowCount(const QModelIndex& parent) const возвращает количество строк в модели, которое соответствует количеству контактов в списке contacts.

2.4.8 Метод columnCount

Метод int PhoneBookModel::columnCount(const QModelIndex& parent) const возвращает количество колонок в модели. В данном случае это фиксированное значение 7.

2.4.9 Метод data

Метод QVariant PhoneBookModel::data(const QModelIndex& index, int role) const в классе PhoneBookModel используется для получения данных из модели в конкретной ячейке таблицы.

Метод data возвращает значение для ячейки данных, расположенной по индексу index. Параметр role указывает на тип запроса В данном случае мы проверяем, что это запрос на отображение данных (Qt::DisplayRole). Затем проверка гарантирует, что метод будет работать только в случае валидного индекса и если запрос касается отображения данных. Если индекс невалиден или роль не Qt::DisplayRole, то метод возвращает пустой QVariant. После извлекаем объект Contact, который находится в строке index.row() в списке contacts. С помощью switch в зависимости от столбца (0-6) метод возвращает соответствующее поле из объекта Contact. Возвращаемые значения для каждого столбца.

2.4.10 Метод headerData

Метод QVariant PhoneBookModel::headerData(int section, Qt::Orientation orientation, int role) const возвращает текст для заголовков столбцов таблицы в модели PhoneBookModel.

role != Qt::DisplayRole метод проверяет, что роль запроса — это отображение данных. Если нет, то возвращает пустое значение (QVariant()). orientation == Qt::Horizontal проверяет, что запрос касается горизонтальных заголовков (столбцов). switch (section) в зависимости от значения section (индекса столбца) возвращается строка с названием столбца

2.4.11 Метод sort

Метод void PhoneBookModel::sort(int column, Qt::SortOrder order) в классе PhoneBookModel сортирует список контактов на основе выбранной колонки и порядка сортировки.

Сначала layoutAboutToBeChanged() уведомляет модель, что данные будут изменены, чтобы она могла обновить отображение, std::sort() сортирует список контактов contacts. В лямбда-функции сравниваются два контакта на основе выбранной колонки и порядка сортировки (возрастание или убывание). column указывает, по какому столбцу будет производиться сортировка, order указывает порядок сортировки (по возрастанию или убыванию) и layoutChanged()после завершения сортировки уведомляет модель, что изменения завершены, и она должна обновить представление.

Ниже приведена блок-схема метода (см. [Рис. 2]).

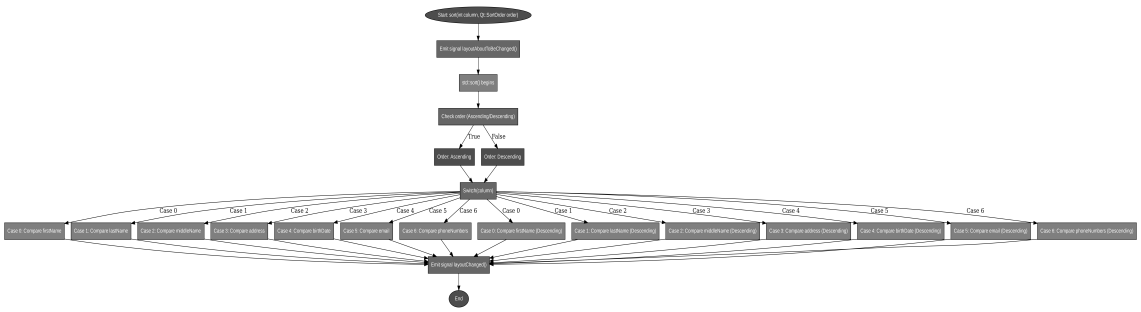

Рис. 2: Блок-схема сортировки

2.5 Класс Contact

Этот класс используется для представления информации о каждом контакте в телефонной книге. Все поля в нем являются публичными, что позволяет их легко изменять и использовать в других частях программы.

2.5.1 Поля класса

QString firstName - хранит имя.
QString lastName - хранит фамилию.
QString middleName - хранит отчество.
QString address - хранит адресс.
QDate birthDate - хранит дату рождения.
QString email - хранит электронную почту.
QStringList phoneNumbers - хранит номера телефонов.

2.5.2 Конструтор

Конструктор с параметрами Contact(QString, QString, QString, QString, QDate, QString, QStringList) позволяет инициализировать объект с конкретными значениями для всех полей.

2.6 validateName

Функция bool validateName(const QString& name) проверяет корректность введенного имени с использованием регулярного выражения (QRegularExpression re(QStringLiteral("∧[A-ZA-ЯЁ][A-Za-zA-Яа-яЁё0-9\\-]*[A-Za-zA-Яа-яЁё0-9]$"))).

∧ - начало строки.

[A-ZA-ЯЁ] - первая заглавная буква (латинская или кириллическая).

[A-Za-zA-Яа-яЁё0-9\\-]* - любые буквы, цифры, дефисы или пробелы.

[A-Za-zA-Яа-яЁё0-9] - последняя буква или цифра.

$ - конец строки.

Применяется метод simplified(), который удаляет незначительные пробелы в начале и в конце строки, а также заменяет несколько пробелов на один.

Регулярное выражение проверяет строку по привденным ранее правилам. Функция возвращает true, если имя соответствует всем условиям (не пустое и соответствует шаблону), и false в противном случае.

2.7 validateEmail

Функция bool validateEmail(const QString& email) проверяет корректность введенной почты с использованием регулярного выражения (QRegularExpression re(R"(∧[A-Z0-9_.]+@[A-Z0-9_.]+$) QRegularExpression::CaseInsensitiveOption)).

∧ — начало строки.

[A-Z0-9_.]+ — одна или более заглавных латинских букв, цифр, точек или подчёркиваний.

@ — обязательный символ '@'. [A-Z0-9_.]+ — доменная часть: буквы, цифры, точки и подчёркивания.

$ — конец строки.

Регулярное выражение проверяет строку по привденным ранее правилам. Функция возвращает true, если почта соответствует всем условиям и состоит из 100 или меньше символов, и false в противном случае.

2.8 validatePhoneNumber

Функция bool validatePhoneNumber(const QString& phone) проверяет корректность введенного номера с использованием регулярного выражения (QRegularExpression

re(R"(∧\+?\d{3,11}$)")).

∧ — начало строки.

\+? — знак '+' может быть в начале строки и является необязательным.

\d{3,11} — ровно 3-11 символов.

$ — конец строки.

Регулярное выражение проверяет строку по привденным ранее правилам(см. Ограничения). Функция возвращает true, если номер соответствует всем условиям, и false в противном случае.

2.9 validateBirthDate

Функция bool validateBirthDate(const QDate& date) возвращает true, если дата рождения меньше или равна текущей, и false в противном случае.

2.10 validateContact

Функция bool validateContact(const Contact& contact). Используются отдельные функции для проверки корректности: если хотя бы одно из этих полей некорректно, переменная isValid становится false. Перебираются все номера телефонов из contact.phoneNumbers, для каждого номера вызывается validatePhoneNumber:если хотя бы один номер некорректен, isValid устанавливается в false, и цикл завершается. Возвращается итоговая переменная isValid, указывающая на корректность всех данных.

3 Тестирование приложения

3.1 Интерфейс приложения

После запуска приложения мы видим основное окно, в котором уже можем работать (см. [Рис. 3]).

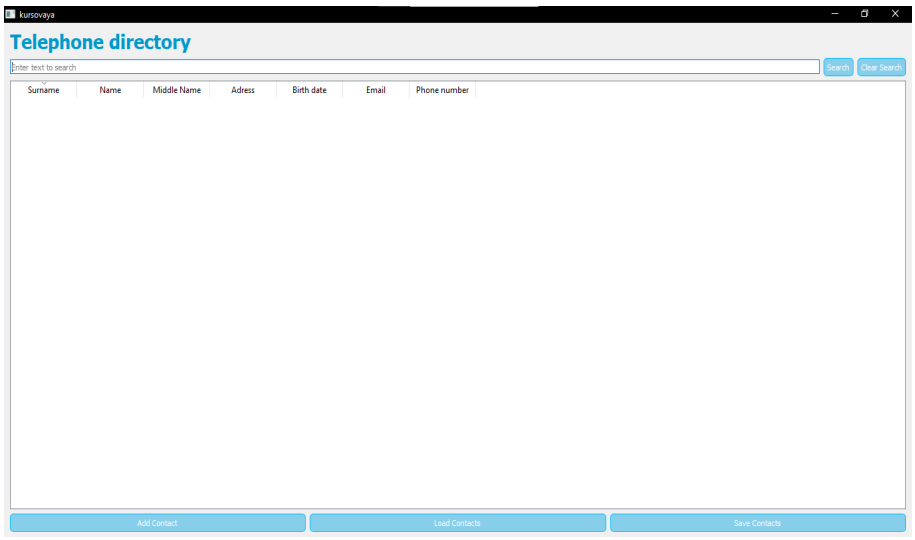

Рис. 3

3.2 Загрузка данных из файла

Добавим в файл *.txt информацию о контактах по правилам, описанным в ограничениях (см. [Рис. 4]).

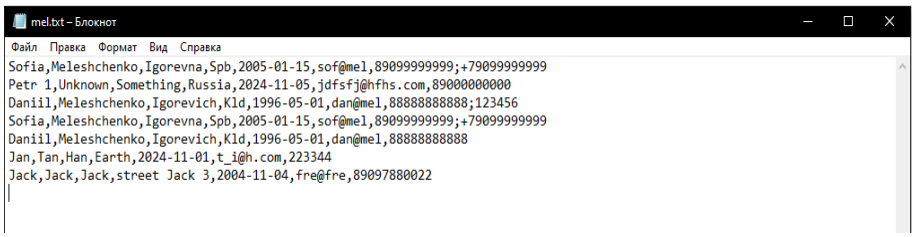

Рис. 4

В самом приложении нажмем на кнопку "Load Contacts"и выберем файл, из которого будем загружать данные (см. [Рис. 5]).

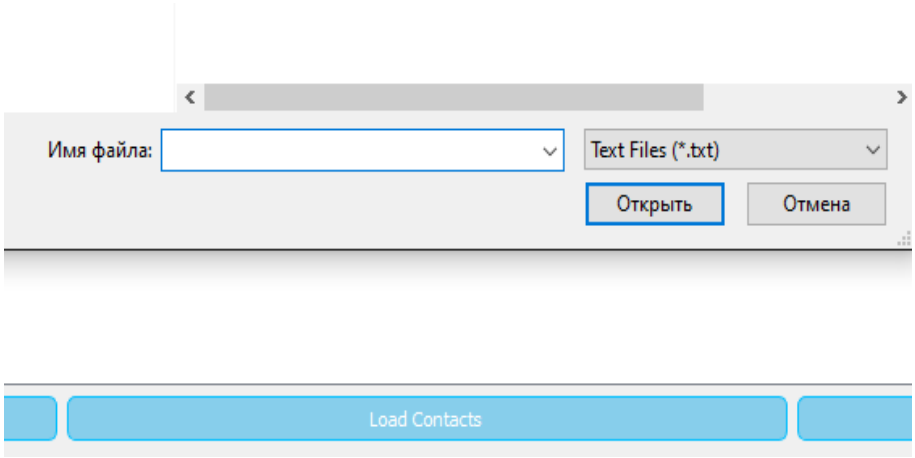

Рис. 5

Данные будут записаны в таблицу (см. [Рис. 6]).

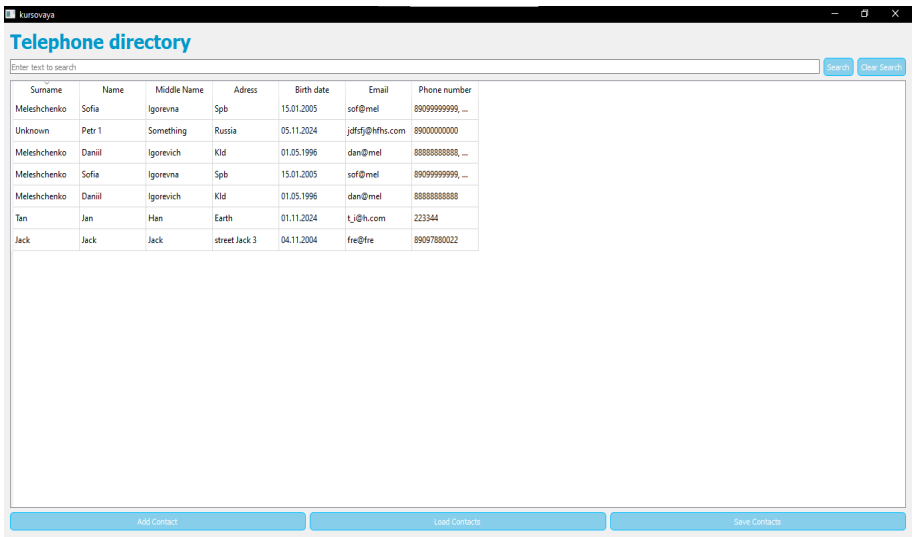

Рис. 6

3.3 Добавление контакта

Для добавления контакта необходимо нажать на кнопку "Add Contact". Появится окно, в которое нужно вводить данные (см. [Рис. 7]).

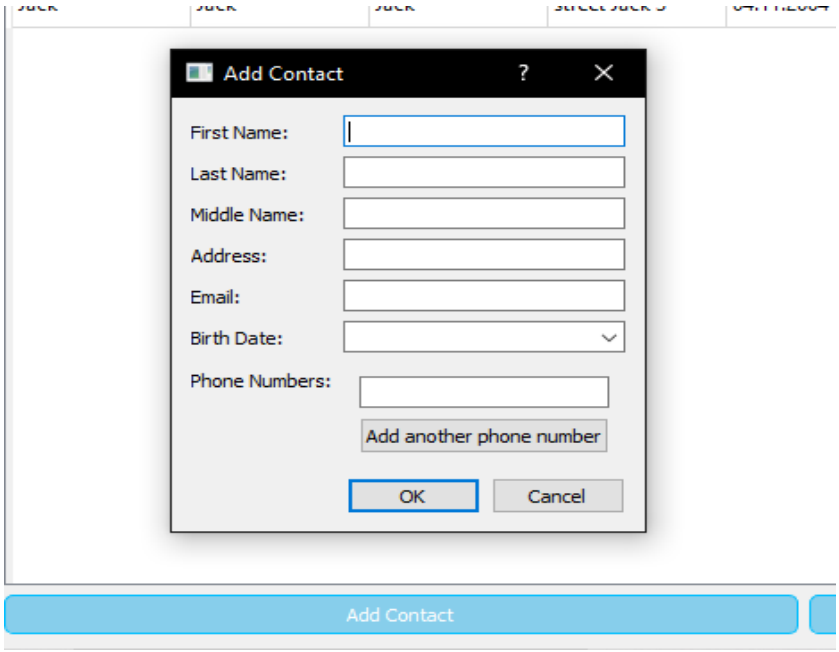

Рис. 7

Введем некорректные данные, не соответствующие ограничениям (см. [Рис. 8]).

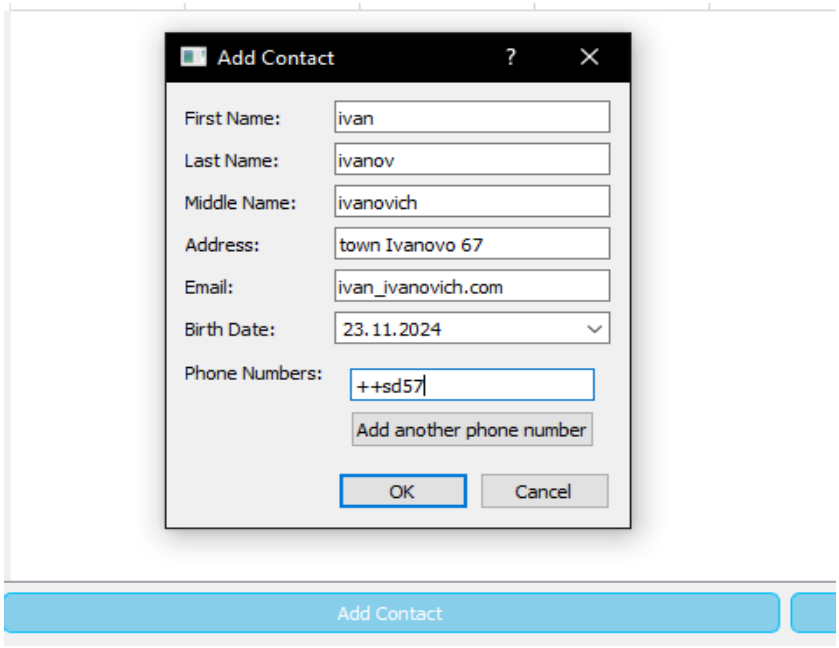

Рис. 8

При нажатии на кнопку "OK"произойдет проверка полей и будет выведено сообщение о том, какие из полей заполнены некорректно (см. [Рис. 9]).

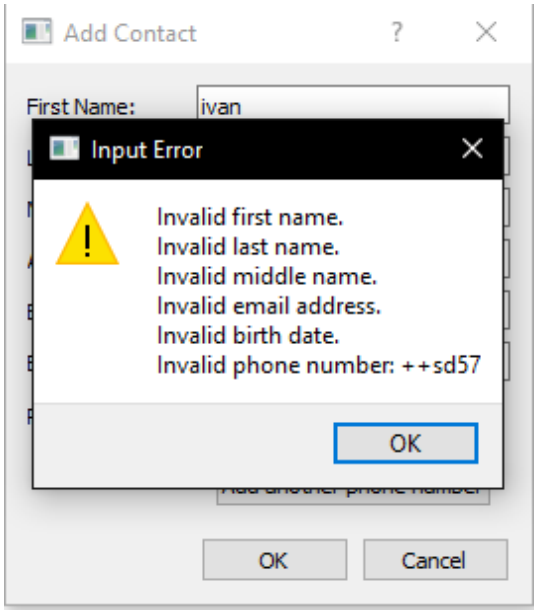

Рис. 9

Введем корректные данные, соответствующие ограничениям (см. [Рис. 10]).

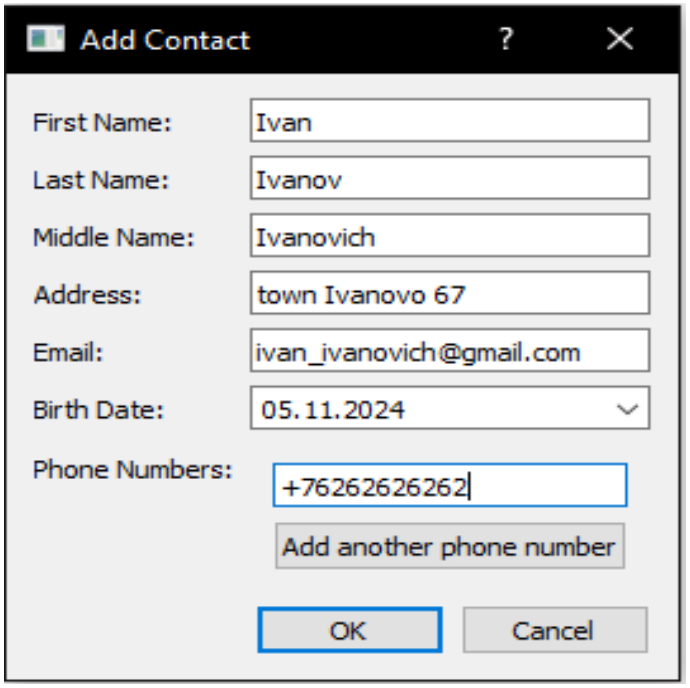

Рис. 10

Также можно добавить несколько номеров с помощью кнопки "Add another phone number"(см. [Рис. 11]).

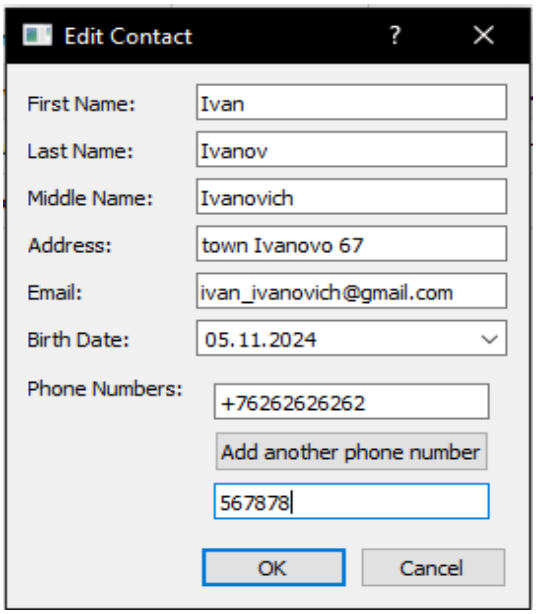

Рис. 11

При нажатии кнопки "OK если все поля корректны, данные будут сохранены в таблице (см. [Рис. 12]).

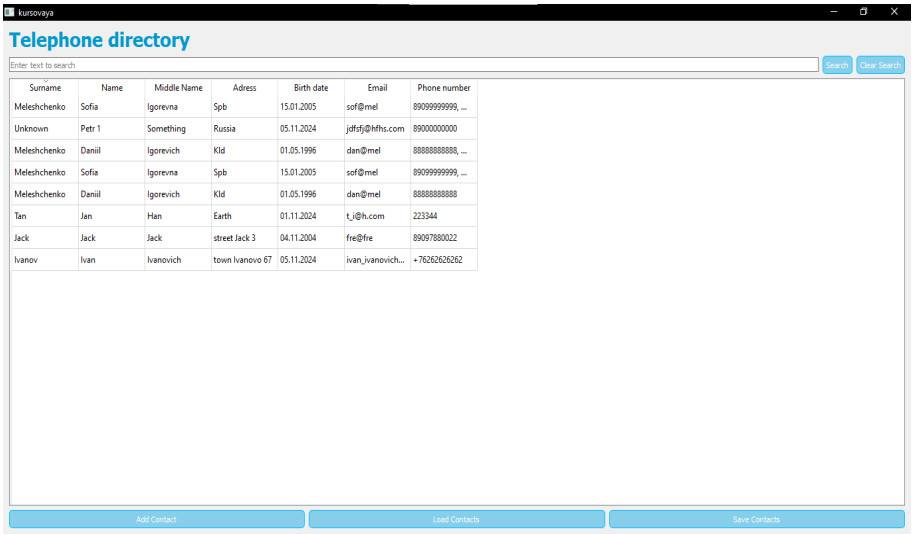

Рис. 12

3.4 Изменение контакта

Чтобы изменить контакт, необходимо правой кнопки мыши нажать на нужное поле, тогда высветится окно предлагающее нам удалить или изменить контакт (см. [Рис. 13]).

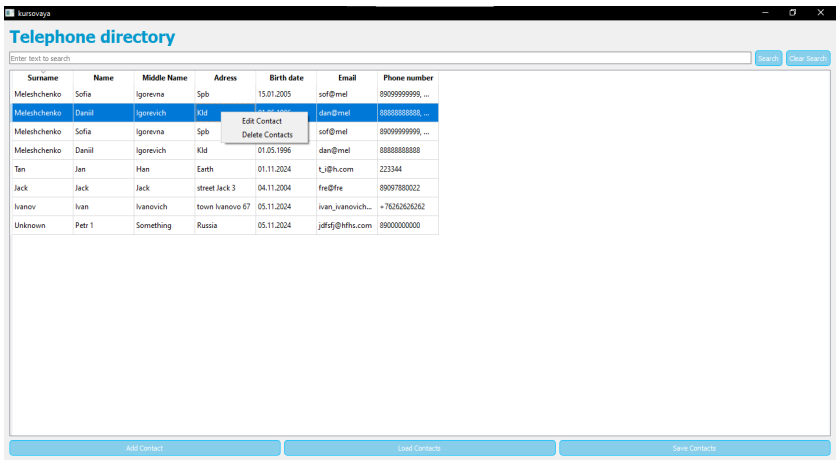

Рис. 13

При нажатии на "Edit Contact появится окно такое же как при создании контакта (см. [Рис. 14]).

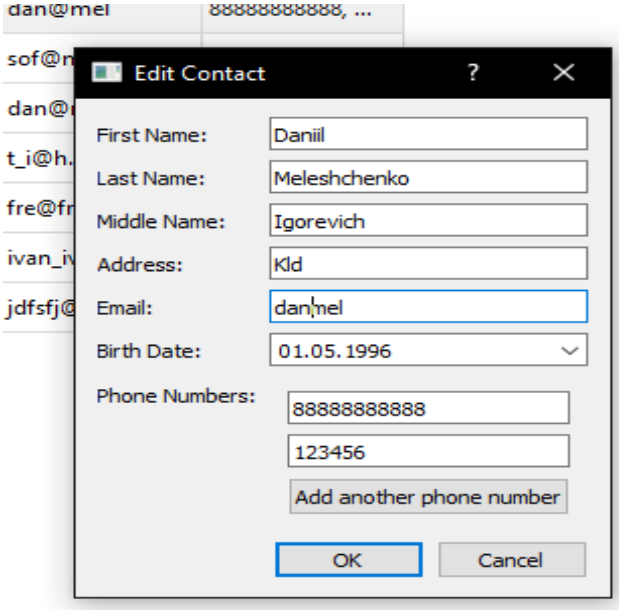

Рис. 14

При некоректном вводе, в данном случае электронный почты и нажатии кнопки "OK"получаем ошибку (см. [Рис. 15]).

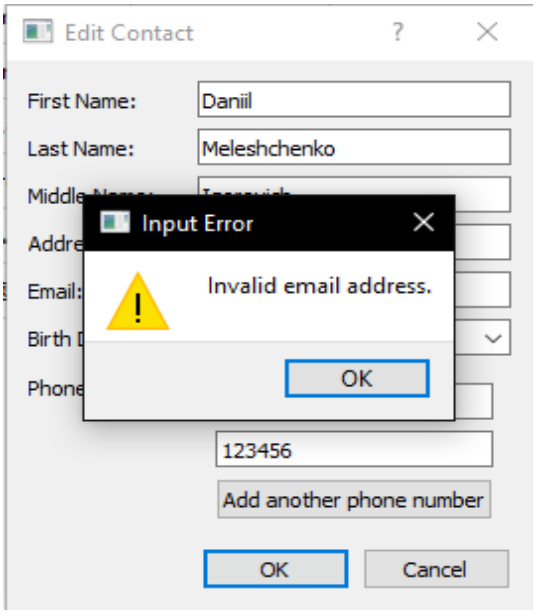

Рис. 15

Если данные введены корректно, то при нажатии на кнопку "OK"они будут сохранены в таблице (см. [Рис. 16]).

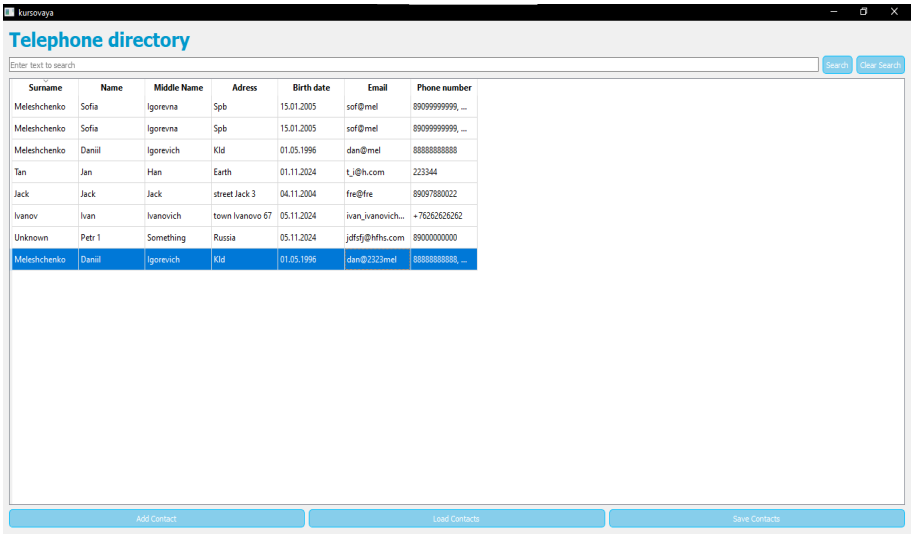

Рис. 16

При выборе нескольких полей, также высветится возможность изменить контакт, но при нажатии на "Edit Contact"будет выведено сообщение об ошибке (см. [Рис. 17]).

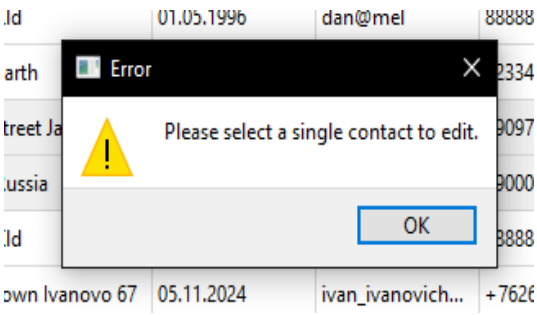

Рис. 17

3.5 Удаление контактов

Для удаления контактов необходимо выбрать один или несколько и нажать на правую кнопку мыши, тогда появится окно выбор действия (см. [Рис. 13]).

После этого нужно подтвердить или отменить действие с помощью кнопок "Yes"/"No"(см. [Рис. 18]).

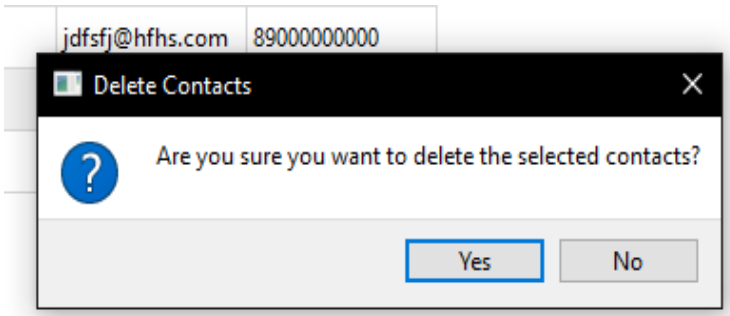

Рис. 18

При нажатии на "Yes"контакт пропадет из таблицы, при нажатии "No"останется.

3.6 Сортировка

Для сортировки необходимо нажать на заголовок столбца, по которому мы хотим сортировать контакты. При нажатии на заголовок, по которому таблица уже отсортирована по возрастанию, происходит сортировка по убыванию.

Ниже приведены примеры сортировки по возрастанию по фамилии (см. [Рис. 19]) и сортировки по убыванию по отчеству (см. [Рис. 20]).

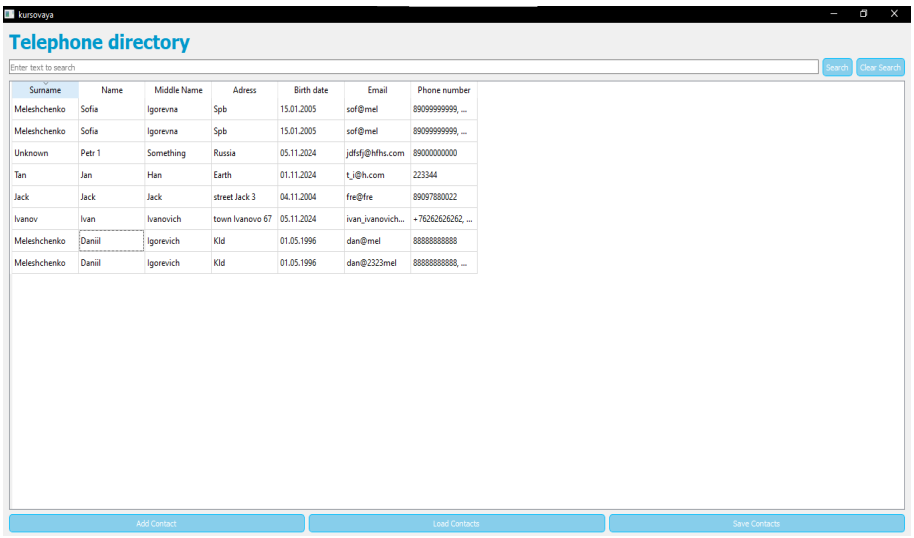

Рис. 19

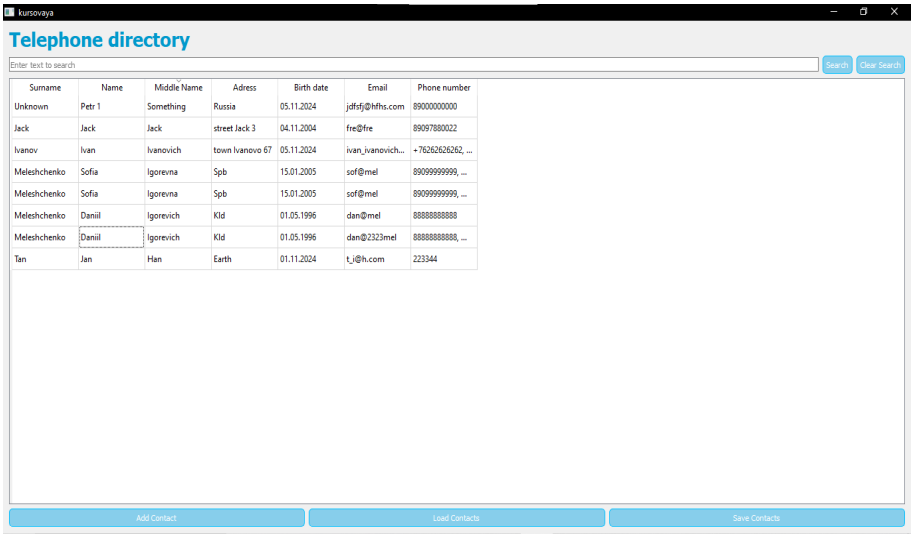

Рис. 20

3.7 Поиск

Для поиска по любому из полей вводим нужные данные в поле и нажимаем "Search". Для примера будем искать по началу строки на "Da"(см. [Рис. 21]).

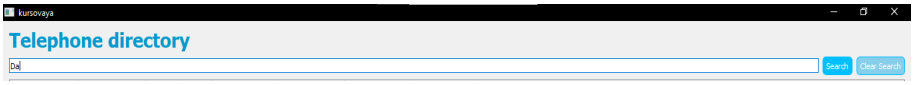

Рис. 21

После нажатия кнопки "Search"будут видны только подходящие варианты (см. [Рис. 22]).

Для того чтобы вернуться к исходной таблице необходимо нажать на кнопку "Clear Search".

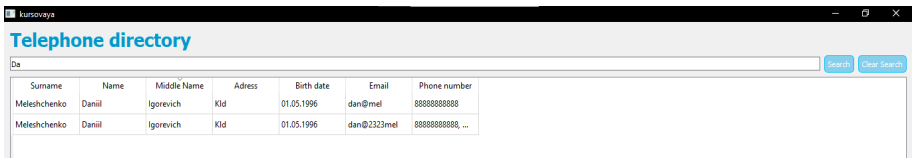

Рис. 22

3.8 Сохранение в файл

После нажатия на кнопку "Save Contacts"высвечивается окно, в котором мы можем создать файл, в который будет перенесена информация о контактах (см. [Рис. 23]).

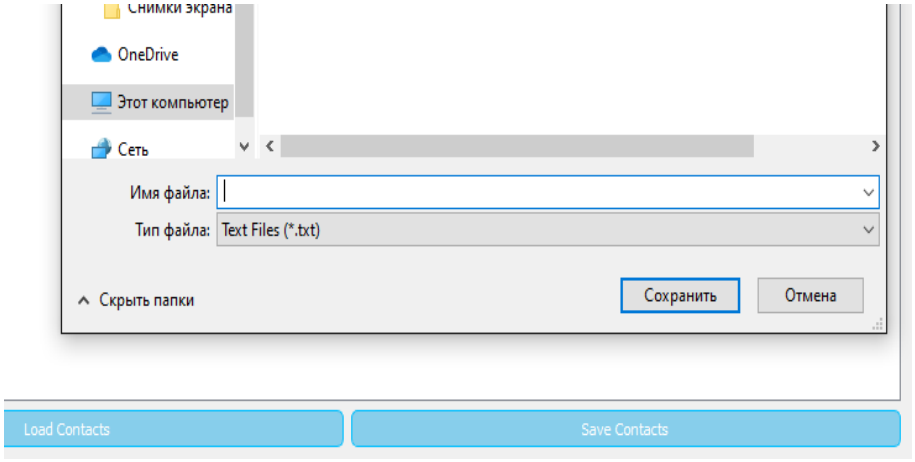

Рис. 23

Ниже приведен файл, в который уже загружена информация (см. [Рис. 24]).

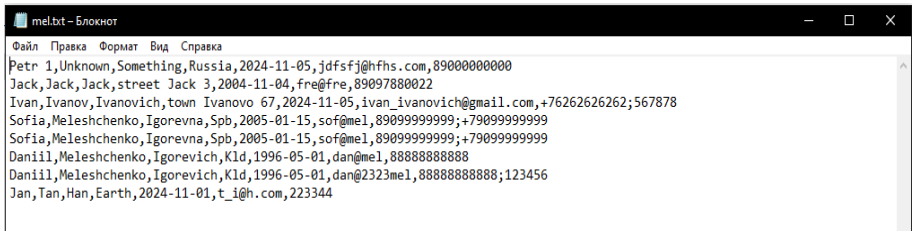

Рис. 24

Заключение

В ходе работы было создано приложение "Телефонный справочник".

Были реализованы все задачи.

1. Хранение контактов. Программа позволяет сохранять контакты, содержащие обязательные поля: имя (QString), фамилия (QString), отчество (QString), адрес (QString), дата рождения (QDate), электронная почта (QString), набор телефонных номеров (QStringList). Для реализации использованы класс Contact (структура данных для хранения информации) и класс PhoneBookModel (контейнер для управления списком контактов).

2. Валидация данных. Все вводимые данные проходят проверку на корректность с использованием регулярных выражений. Реализовано функциями validateName, validateEmail, validatePhoneNumber, validateBirthDate и validateContact.

3. Добавление новых контактов. Добавление контактов осуществляется с помощью метода MainWindow::addContact.

4. Редактирование существующих записей. Метод

MainWindow::editSelectedContact позволяет изменять данные уже существующих контактов.

5. Удаление контактов. Для удаления одного или нескольких контактов реализован метод MainWindow::deleteSelectedContacts с поддержкой множественного выбора.

6. Поиск по контактов. Функция поиска реализована методом

MainWindow::searchContacts, позволяющим поиск по подстроке или строке в любом поле контакта.

7. Сортировка контактов. Контакты можно отсортировать по любому полю в порядке возрастания или убывания с помощью метода PhoneBookModel::sort.

8. Сохранение данных в файл. Метод PhoneBookModel::saveToFile позволяет сохранить контакты в текстовый файл *.txt. Данные сохраняются в структурированном формате для их последующего использования (поля через запятую, номера телефонов через точку с запятой).

9. Загрузка данных из файла. Восстановление данных реализовано методом

PhoneBookModel::loadFromFile.

В ходе разработки приложения были изучены и освоены следующие аспекты: работа с библиотеками Qt, работа с файлами, использование регулярных выражений для проверки корректности полей, сортировка данных.

QtCreator 5.12.10.
Visual Studio 2022.
Язык программирования: C++.
Компилятор: MSVC.

Список литературы

[1] Т. А. Павловская. C/C++ Программирование на языке высокого уровня. Питер 2004.
(дата обращения: 21.10.2024)

[2] Макс Шлее. Qt5.10 Профессиональное программирование на C++. 2022
(дата обращения: 10.11.2024)

[3] Qt.
// URL: https://blog.skillfactory.ru/glossary/qt
(дата обращения: 20.11.2024)

[4] Qt documentation.
// URL: https://doc.qt.io/
(дата обращения: 13.11.2024)

[5] Особенности Qt: слоты и сигналы, описание QObject и QApplication, виды окон и т.д.
//URL: http://cppstudio.com/post/11167/
(дата обращения: 13.11.2024)

Приложение A. Main.cpp

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

int main(int argc, char* argv[]) {
    QApplication app(argc, argv);
    MainWindow window;
    window.show();
    return app.exec();
}
```

Приложение Б. contact.h

```
#ifndef CONTACT_H
#define CONTACT_H
#include <QString>
#include <QDate>

class Contact {
public:
    Contact() = default;
    Contact(QString firstName, QString lastName, QString middleName, QString address,
        QDate birthDate, QString email, QStringList phoneNumbers)
        : firstName(firstName), lastName(lastName), middleName(middleName),
        address(address), birthDate(birthDate), email(email), phoneNumbers(phoneNumbers) {}

    QString firstName;
    QString lastName;
    QString middleName;
    QString address;
    QDate birthDate;
    QString email;
    QStringList phoneNumbers;
};

#endif
```

Приложение B. validation.h

```
#ifndef VALIDATION_H
#define VALIDATION_H
#include <QString>
#include <QDate>

bool validateName(const QString& name);
bool validateEmail(const QString& email);
bool validatePhoneNumber(const QString& phone);
bool validateBirthDate(const QDate& date);

#endif
```

Приложение Г. validation.cpp

```
#include "validation.h"
#include <QRegularExpression>
#include <QDate>
#include "contact.h"

bool validateName(const QString& name) {
    QString trimmedName = name.simplified(); // Убираем незначимые пробелы
    // ^ - начало строки
    // [A-ZA-ЯЁ] - первая заглавная буква (латинская или кириллическая)
    // [A-Za-zA-Яа-яЁё0-9\\- ]* - любые буквы, цифры, дефисы или пробелы
    // [A-Za-zA-Яа-яЁё0-9] - последняя буква или цифра
    // $ - конец строки
    QRegularExpression re(QStringLiteral("^[A-ZA-ЯЁ][A-Za-zA-Яа-яЁё0-9\\- ]*[A-Za-zA-Яа-яЁё0-9]$"));
    return !trimmedName.isEmpty() && re.match(trimmedName).hasMatch();
}

bool validatePhoneNumber(const QString& phone) {
       // ^         - начало строки.
       // \+?       - знак '+' может быть в начале строки и является необязательным.
       // \d{3,11} - ровно 3-11 символов.
       // $         - конец строки.
    QRegularExpression re(R"(^\+?\d{3,11}$)");
    return re.match(phone.simplified()).hasMatch();
}

bool validateBirthDate(const QDate& date) {
    return date.isValid() && date <= QDate::currentDate();
}

bool validateEmail(const QString& email) {
    // ^               - начало строки
    // [A-Z0-9_.]+     - одна или более заглавных латинских букв, цифр, точек или подчёркиваний
    // @               - обязательный символ '@'
    // [A-Z0-9_.]+     - доменная часть: буквы, цифры, точки и подчёркивания
    // $               - конец строки
    QRegularExpression re(R"(^[A-Z0-9_.]+@[A-Z0-9_.]+$)", QRegularExpression::CaseInsensitiveOption);
    return email.simplified().length() <= 100 && re.match(email.simplified()).hasMatch();
}


bool validateContact(const Contact& contact) {
    bool isValid = validateName(contact.firstName) &&
        validateName(contact.lastName) &&
        validateName(contact.middleName) &&
        validateEmail(contact.email) &&
        validateBirthDate(contact.birthDate);

    for (const QString& phoneNumber : contact.phoneNumbers) {
        if (!validatePhoneNumber(phoneNumber)) {
            isValid = false;
            break;
        }
    }

    return isValid;
}
```

Приложение Д. mainwindow.h

```
 #ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QPushButton>
#include <QLineEdit>
#include <QTableView>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QLabel>
#include <QInputDialog>
#include "phonebookmodel.h"
#include "contact.h"

class MainWindow : public QMainWindow {
    Q_OBJECT
    PhoneBookModel* phoneBookModel;
    QTableView* tableView;
    QMenu* contextMenu;
    QLineEdit* searchLineEdit;
    QPushButton* addButton;
    QPushButton* loadButton;
    QPushButton* saveButton;
    QPushButton* searchButton;
    QPushButton* clearSearchButton;

    void setupContextMenu();
    void editSelectedContact();
    void deleteSelectedContacts();
    void setupUI();
    void setupConnections();
    Contact getContactFromUser(const QString& title, const Contact& defaultContact = Contact());

private slots:
    void addContact();
    void loadContacts();
    void saveContacts();
    void searchContacts();
    void clearSearch();
    QHBoxLayout* searchLayout();

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

};

#endif
```

Приложение Е. mainwindow.cpp

```
 #include "mainwindow.h"
#include <QFileDialog>
#include <QMessageBox>
#include <QInputDialog>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QDateEdit>
#include <QApplication>
#include <QDialog>
#include <QFormLayout>
#include <QDialogButtonBox>
#include <QDate>
#include "validation.h"
#include "QMenu"


MainWindow::MainWindow(QWidget* parent)
    : QMainWindow(parent), phoneBookModel(new PhoneBookModel(this)) {

    setupUI();
    setupConnections();

    foreach(QPushButton * button, this->findChildren<QPushButton*>()) {
        button->setStyleSheet(
            "QPushButton {"
            "   background-color: #87CEEB;"
            "   color: white;"
            "   border: 1px solid #00BFFF;"
            "   border-radius: 5px;"
            "   padding: 5px;"
            "}"
            "QPushButton:hover {"
            "   background-color: #00BFFF;"
            "}"
            "QPushButton:pressed {"
            "   background-color: #4682B4;"
            "}"
        );
    }

}

void MainWindow::setupUI() {
    QWidget* centralWidget = new QWidget(this);
    setCentralWidget(centralWidget);

    addButton = new QPushButton("Add Contact");
    loadButton = new QPushButton("Load Contacts");
    saveButton = new QPushButton("Save Contacts");
    searchButton = new QPushButton("Search");
    clearSearchButton = new QPushButton("Clear Search");

    searchLineEdit = new QLineEdit();
    searchLineEdit->setPlaceholderText("Enter text to search");

    tableView = new QTableView();
    tableView->setModel(phoneBookModel);
    tableView->setSortingEnabled(true);
```

```
    tableView->setContextMenuPolicy(Qt::CustomContextMenu);

    setupContextMenu();

    QLabel* titleLabel = new QLabel("Telephone directory"); //заголовок
    QFont font = titleLabel->font();
    font.setPointSize(20);
    font.setBold(true);
    titleLabel->setFont(font);
    titleLabel->setStyleSheet("color: #0099cc;");

    tableView->setSelectionBehavior(QAbstractItemView::SelectRows);
    tableView->setEditTriggers(QAbstractItemView::NoEditTriggers);
    tableView->setSelectionMode(QAbstractItemView::ExtendedSelection);

    QVBoxLayout* mainLayout = new QVBoxLayout();
    mainLayout->addWidget(titleLabel);
    mainLayout->addLayout(searchLayout());
    mainLayout->addWidget(tableView);

    QHBoxLayout* buttonLayout = new QHBoxLayout();
    buttonLayout->addWidget(addButton);
    buttonLayout->addWidget(loadButton);
    buttonLayout->addWidget(saveButton);

    mainLayout->addLayout(buttonLayout);
    centralWidget->setLayout(mainLayout);
}

void MainWindow::setupConnections() {
    connect(addButton, &QPushButton::clicked, this, &MainWindow::addContact);
    connect(loadButton, &QPushButton::clicked, this, &MainWindow::loadContacts);
    connect(saveButton, &QPushButton::clicked, this, &MainWindow::saveContacts);
    connect(searchButton, &QPushButton::clicked, this, &MainWindow::searchContacts);
    connect(clearSearchButton, &QPushButton::clicked, this, &MainWindow::clearSearch);
}

QHBoxLayout* MainWindow::searchLayout() {
    QHBoxLayout* layout = new QHBoxLayout();
    layout->addWidget(searchLineEdit);
    layout->addWidget(searchButton);
    layout->addWidget(clearSearchButton);
    return layout;
};

Contact MainWindow::getContactFromUser(const QString& title, const Contact& defaultContact) {
    QDialog dialog(this);
    dialog.setWindowTitle(title);

    QFormLayout* formLayout = new QFormLayout(&dialog);
    QLineEdit* firstNameEdit = new QLineEdit(defaultContact.firstName);
    QLineEdit* lastNameEdit = new QLineEdit(defaultContact.lastName);
    QLineEdit* middleNameEdit = new QLineEdit(defaultContact.middleName);
    QLineEdit* addressEdit = new QLineEdit(defaultContact.address);
    QLineEdit* emailEdit = new QLineEdit(defaultContact.email);

    QDateEdit* birthDateEdit = new QDateEdit(defaultContact.birthDate);
    birthDateEdit->setCalendarPopup(true);
    birthDateEdit->setDisplayFormat("dd.MM.yyyy");
```

```
QVBoxLayout* phoneNumbersLayout = new QVBoxLayout();
QList<QLineEdit*> phoneNumberEdits;

for (const QString& phoneNumber : defaultContact.phoneNumbers) {
    QLineEdit* phoneEdit = new QLineEdit(phoneNumber);
    phoneNumberEdits.append(phoneEdit);
    phoneNumbersLayout->addWidget(phoneEdit);
}
if (phoneNumberEdits.isEmpty()) {
    QLineEdit* initialPhoneEdit = new QLineEdit();
    phoneNumberEdits.append(initialPhoneEdit);
    phoneNumbersLayout->addWidget(initialPhoneEdit);
}

QPushButton* addPhoneButton = new QPushButton("Add another phone number");
phoneNumbersLayout->addWidget(addPhoneButton);

connect(addPhoneButton, &QPushButton::clicked, this, [&]() {
    QLineEdit* newPhoneEdit = new QLineEdit();
    phoneNumberEdits.append(newPhoneEdit);
    phoneNumbersLayout->addWidget(newPhoneEdit);
    });

formLayout->addRow("First Name:", firstNameEdit);
formLayout->addRow("Last Name:", lastNameEdit);
formLayout->addRow("Middle Name:", middleNameEdit);
formLayout->addRow("Address:", addressEdit);
formLayout->addRow("Email:", emailEdit);
formLayout->addRow("Birth Date:", birthDateEdit);

QWidget* phoneNumbersWidget = new QWidget();
phoneNumbersWidget->setLayout(phoneNumbersLayout);
formLayout->addRow("Phone Numbers:", phoneNumbersWidget);

QDialogButtonBox* buttonBox = new QDialogButtonBox(QDialogButtonBox::Ok \
| QDialogButtonBox::Cancel);
formLayout->addWidget(buttonBox);

connect(buttonBox, &QDialogButtonBox::accepted, &dialog, [&]() {
    QStringList errorMessages;

    if (!validateName(firstNameEdit->text())) errorMessages.append("Invalid first name.");
    if (!validateName(lastNameEdit->text())) errorMessages.append("Invalid last name.");
    if (!validateName(middleNameEdit->text())) errorMessages.append("Invalid middle name.");
    if (!validateEmail(emailEdit->text())) errorMessages.append("Invalid email address.");
    if (!validateBirthDate(birthDateEdit->date())) errorMessages.append("Invalid birth date.");

    for (QLineEdit* phoneEdit : phoneNumberEdits) {
        QString phoneNumber = phoneEdit->text().trimmed();
        if (!phoneNumber.isEmpty() && !validatePhoneNumber(phoneNumber))
            errorMessages.append("Invalid phone number: " + phoneNumber);
    }

    if (!errorMessages.isEmpty()) {
        QMessageBox::warning(&dialog, "Input Error", errorMessages.join("\n"));
        return;
    }
    dialog.accept();
    });
```

```
    connect(buttonBox, &QDialogButtonBox::rejected, &dialog, &QDialog::reject);

    if (dialog.exec() == QDialog::Accepted) {
        Contact contact;
        contact.firstName = firstNameEdit->text();
        contact.lastName = lastNameEdit->text();
        contact.middleName = middleNameEdit->text();
        contact.address = addressEdit->text();
        contact.email = emailEdit->text();
        contact.birthDate = birthDateEdit->date();

        QStringList phoneNumbers;
        for (QLineEdit* phoneEdit : phoneNumberEdits) {
            QString phoneNumber = phoneEdit->text().trimmed();
            if (!phoneNumber.isEmpty()) phoneNumbers.append(phoneNumber);
        }
        contact.phoneNumbers = phoneNumbers;
        return contact;
    }
    return defaultContact;
}

void MainWindow::addContact() {
    Contact contact = getContactFromUser("Add Contact");
    phoneBookModel->addContact(contact);
}

void MainWindow::loadContacts() {
    QString filename = QFileDialog::getOpenFileName(this, "Load File", "", "Text Files (*.txt)");
    if (!filename.isEmpty()) phoneBookModel->loadFromFile(filename);
}

void MainWindow::saveContacts() {
    QString filename = QFileDialog::getSaveFileName(this, "Save File", "", "Text Files (*.txt)");
    if (!filename.isEmpty()) phoneBookModel->saveToFile(filename);
}

void MainWindow::searchContacts() {
    QString searchTerm = searchLineEdit->text();
    for (int row = 0; row < phoneBookModel->rowCount(); ++row) {
        bool match = false;
        for (int col = 0; col < phoneBookModel->columnCount(); ++col) {
            QModelIndex index = phoneBookModel->index(row, col);
            if (phoneBookModel->data(index).toString().contains(searchTerm, Qt::CaseInsensitive)) {
                match = true;
                break;
            }
        }
        tableView->setRowHidden(row, !match);
    }
}

void MainWindow::clearSearch() {
    searchLineEdit->clear();
    for (int row = 0; row < phoneBookModel->rowCount(); ++row) {
        tableView->setRowHidden(row, false);
    }
}

void MainWindow::setupContextMenu() {
```

```
    contextMenu = new QMenu(this);

    QAction* editAction = new QAction("Edit Contact", this);
    QAction* deleteAction = new QAction("Delete Contacts", this);

    connect(editAction, &QAction::triggered, this, &MainWindow::editSelectedContact);
    connect(deleteAction, &QAction::triggered, this, &MainWindow::deleteSelectedContacts);

    contextMenu->addAction(editAction);
    contextMenu->addAction(deleteAction);

    connect(tableView, &QTableView::customContextMenuRequested, this, [this](const QPoint& pos) {
        QModelIndex index = tableView->indexAt(pos);
        if (index.isValid())  contextMenu->exec(tableView->viewport()->mapToGlobal(pos));
        });
}


// Изменение выбранного контакта
void MainWindow::editSelectedContact() {
    QModelIndexList selectedRows = tableView->selectionModel()->selectedRows();

    if (selectedRows.size() == 1) {
        int selectedRow = selectedRows.first().row();
        Contact currentContact = phoneBookModel->contacts.at(selectedRow);
        Contact editedContact = getContactFromUser("Edit Contact", currentContact);

        phoneBookModel->removeContact(selectedRow);
        phoneBookModel->addContact(editedContact);
    }
    else  QMessageBox::warning(this, "Error", "Please select a single contact to edit.");
}

void MainWindow::deleteSelectedContacts() {
    QModelIndexList selectedRows = tableView->selectionModel()->selectedRows();

    if (selectedRows.isEmpty()) {
        QMessageBox::warning(this, "Error", "No contacts selected for deletion.");
        return;
    }

    QMessageBox::StandardButton reply;
    reply = QMessageBox::question(this, "Delete Contacts", \
    "Are you sure you want to delete the selected contacts?",
        QMessageBox::Yes | QMessageBox::No);

    if (reply == QMessageBox::Yes) {
        std::sort(selectedRows.begin(), selectedRows.end(), [](const QModelIndex& a,\
        const QModelIndex& b) {
            return a.row() > b.row();});

        for (const QModelIndex& index : selectedRows) phoneBookModel->removeContact(index.row());
    }
}
```

Приложение Ж. phonebookmodel.h

```
 #ifndef PHONEBOOKMODEL_H
#define PHONEBOOKMODEL_H
#include <QAbstractTableModel>
#include "contact.h"

class PhoneBookModel : public QAbstractTableModel {
    Q_OBJECT
public:
    explicit PhoneBookModel(QObject* parent)
        : QAbstractTableModel(parent) {};
    QList<Contact> contacts;
    QVariant data(const QModelIndex& index, int role = Qt::DisplayRole) const override;
    QVariant headerData(int section, Qt::Orientation orientation, int role = Qt::DisplayRole)\
    const override;
    void addContact(const Contact& contact);
    void removeContact(int row);
    int rowCount(const QModelIndex& parent = QModelIndex()) const override;
    int columnCount(const QModelIndex& parent = QModelIndex()) const override;
    void saveToFile(const QString& filename);
    void loadFromFile(const QString& filename);
    void sort(int column, Qt::SortOrder order);

};
#endif
```

Приложение 3. phonebookmodel.cpp

```
#include "phonebookmodel.h"
#include "contact.h"
#include "validation.h"
#include <QFile>
#include <QTextStream>
#include <QMessageBox>

void PhoneBookModel::saveToFile(const QString& filename) {
    QFile file(filename);
    if (file.open(QIODevice::WriteOnly)) {
        QTextStream out(&file);
        for (const auto& contact : contacts) {
            out << contact.firstName << "," << contact.lastName << ","
                << contact.middleName << "," << contact.address << ","
                << contact.birthDate.toString(Qt::ISODate) << ","
                << contact.email << "," << contact.phoneNumbers.join(";") << "\n";
        }
        file.close();
    }
}

void PhoneBookModel::loadFromFile(const QString& filename) {
    QFile file(filename);
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) return;

    QTextStream in(&file);
    QList<Contact> newContacts;

    while (!in.atEnd()) {
        QStringList data = in.readLine().split(",");
        if (data.size() >= 7) {
            Contact contact(
                data[0].simplified(),
                data[1].simplified(),
                data[2].simplified(),
                data[3].simplified(),
                QDate::fromString(data[4], Qt::ISODate),
                data[5].simplified(),
                data[6].split(";")
            );
            newContacts.append(contact);
        }
    }
    file.close();
    beginResetModel();
    contacts = newContacts;
    endResetModel();
}

void PhoneBookModel::addContact(const Contact& contact) {
    if (!validateName(contact.firstName) || !validateName(contact.lastName) || \
    !validateName(contact.middleName)\
        || !validateEmail(contact.email) || !validateBirthDate(contact.birthDate)) return;
    for (const QString& phoneNumber : contact.phoneNumbers) {
        if (!validatePhoneNumber(phoneNumber)) return;
    }
    beginInsertRows(QModelIndex(), contacts.size(), contacts.size());
    contacts.append(contact);
```

```
    endInsertRows();
}

void PhoneBookModel::removeContact(int row) {
    if (row < 0 || row >= contacts.size()) return;
    beginRemoveRows(QModelIndex(), row, row);
    contacts.removeAt(row);
    endRemoveRows();
}

int PhoneBookModel::rowCount(const QModelIndex& parent) const {
    return contacts.size();
}

int PhoneBookModel::columnCount(const QModelIndex& parent) const {
    return 7;
}

QVariant PhoneBookModel::data(const QModelIndex& index, int role) const {
    if (!index.isValid() || role != Qt::DisplayRole) return QVariant();

    const Contact& contact = contacts.at(index.row());
    switch (index.column()) {
    case 0: return contact.lastName;
    case 1: return contact.firstName;
    case 2: return contact.middleName;
    case 3: return contact.address;
    case 4: return contact.birthDate.toString("dd.MM.yyyy");
    case 5: return contact.email;
    case 6: return contact.phoneNumbers.join(", ");
    default: return QVariant();
    }
}

QVariant PhoneBookModel::headerData(int section, Qt::Orientation orientation, int role) const {
    if (role != Qt::DisplayRole)  return QVariant();

    if (orientation == Qt::Horizontal) {
        switch (section) {
        case 0: return "Surname";
        case 1: return "Name";
        case 2: return "Middle Name";
        case 3: return "Adress";
        case 4: return "Birth date";
        case 5: return "Email";
        case 6: return "Phone number";
        default: return QVariant();
        }
    }
    return QVariant();
}

void PhoneBookModel::sort(int column, Qt::SortOrder order) {
    emit layoutAboutToBeChanged();
    std::sort(contacts.begin(), contacts.end(), [column, order](const Contact& a, \
    const Contact& b) {
        if (order == Qt::AscendingOrder) {
            switch (column) {
            case 0: return a.firstName < b.firstName;
            case 1: return a.lastName < b.lastName;
```

```
            case 2: return a.middleName < b.middleName;
            case 3: return a.address < b.address;
            case 4: return a.birthDate < b.birthDate;
            case 5: return a.email < b.email;
            case 6: return a.phoneNumbers.join(", ") < b.phoneNumbers.join(", ");
            default: return false;
            }
        }
        else {
            switch (column) {
            case 0: return a.firstName > b.firstName;
            case 1: return a.lastName > b.lastName;
            case 2: return a.middleName > b.middleName;
            case 3: return a.address > b.address;
            case 4: return a.birthDate > b.birthDate;
            case 5: return a.email > b.email;
            case 6: return a.phoneNumbers.join(", ") > b.phoneNumbers.join(", ");
            default: return false;
            }
        }
        });
    emit layoutChanged();
}
```