ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»
Институт компьютерных наук и кибербезопасности
Высшая школа технологий искусственного интеллекта
02.03.03 Математическое обеспечение и администрирование информационных систем

Приложение «Телефонрый справочник» с интерфейсами командной строки и графическим интерфейсом пользователя, использующее фреймворк Qt.

Дисциплина: Объектно-ориентированное программирование

студент гр. 5130203/40001 Чиканге М.
преподаватель: Йовановски Ненад

« » 2025 г.

Санкт-Петербург.

2025 г.

Оглавление

1. ВВЕДЕНИЕ

Телефонный справочник является типичным примером прикладной информационной системы, предназначенной для хранения, поиска и управления персональными контактными данными. В рамках данной работы разрабатывается приложение «Телефонный справочник» с использованием библиотеки Qt, ориентированное на практическую реализацию полного цикла управления контактами: создание, просмотр, редактирование, удаление, сортировка и поиск записей.

Цель проекта спроектировать и реализовать приложение, которое обеспечивает удобный графический интерфейс, корректную обработку пользовательского ввода и надёжное хранение данных. В работе последовательно решаются несколько задач, каждая из которых оформляется в отдельной ветке репозитория (branch) с созданием Pull Request и ведением истории разработки. Такой подход позволяет продемонстрировать этапность разработки, аккуратную структуру кода и контроль качества внесённых изменений.

Приложение должно поддерживать хранение обязательных полей контакта: фамилия, имя, отчество, адрес, дата рождения, e-mail, а также телефонные номера (рабочий, домашний, служебный) в любом количестве. При этом для создания контакта обязательными являются фамилия, имя, e-mail и минимум один номер телефона. Особое внимание уделяется корректности вводимых данных: все поля должны проходить валидацию с использованием регулярных выражений, а незначащие пробелы удаляться. Для ФИО требуется соблюдение формата: допускаются буквы и цифры различных алфавитов, дефис и пробел, но строка должна начинаться с буквы и не может начинаться или заканчиваться дефисом. Для телефонных номеров предусмотрены несколько допустимых форматов записи (включая варианты с кодом страны, скобками и дефисами). Дата рождения должна быть меньше текущей даты, с учётом корректного количества дней в месяцах и високосных годов. E-mail должен содержать имя пользователя и домен, разделённые символом «@», и состоять из латинских букв и цифр; все пробелы, включая пробелы вокруг «@», удаляются.

Хранение данных реализуется поэтапно. На первом этапе создаётся консольная версия приложения без Qt с сохранением контактов в файл. Затем добавляется графический интерфейс на Qt, при этом файловые операции чтения и записи выполняются с использованием QFile. На следующем этапе, не удаляя файловое хранение, реализуется дополнительная возможность хранения данных в базе данных PostgreSQL, что позволяет сравнить подходы к персистентности и обеспечить расширяемость системы. Дополнительно предусматривается углублённая задача по оптимизации работы с памятью: переопределение операторов new, copy и move для класса контакта с анализом числа копирований и созданных объектов и, по возможности, уменьшением этих затрат.

Итогом работы является приложение, демонстрирующее корректную работу всех ключевых функций (добавление, удаление, редактирование, сортировка, поиск), устойчивость к некорректному вводу, а также корректность чтения и записи данных как в файл, так и в базу данных.

2. ПОСТАНОВКА ЗАДАЧИ

Необходимо разработать приложение «Телефонный справочник» на языке C++ с последующим расширением с использованием библиотеки Qt. Проект выполняется поэтапно (задачи 0–4), при этом каждый этап должен оформляться в отдельной ветке репозитория с Pull Request, который не закрывается до принятия преподавателем.

В рамках текущей реализации (консольная версия, без Qt) требуется обеспечить полноценное управление контактами и корректную работу с постоянным хранилищем в файле. Реализация должна поддерживать следующие функции и ограничения:

2.1. Модель данных контакта

Контакт представляет собой структурированную запись, включающую:

- имя (firstName);
- фамилию (lastName);
- отчество (middleName) — опционально;
- e-mail;
- адрес — опционально;
- дату рождения — опционально;
- телефонные номера в разрезе категорий: рабочий, домашний, служебный (в реализации представлены тремя полями).

Обязательные поля для создания контакта в реализованной версии:

- имя;
- фамилия;
- e-mail;
- минимум один телефонный номер (рабочий, домашний или служебный).

2.2. Функциональные требования (консольный интерфейс)

Приложение должно предоставлять пользователю меню и обеспечивать операции:

1. добавление контакта (Create contact);
2. поиск контакта (Search contact) по одному выбранному ключу;
3. редактирование контакта (Edit contact) с возможностью изменения каждого поля по отдельности;
4. удаление контакта (Delete contact) с подтверждением;
5. вывод списка контактов с сортировкой (List contacts / sorted).

Поиск в реализованной версии выполняется по одному из следующих полей (выбор метода из меню):

- имя;
- фамилия;
- рабочий телефон;
- домашний телефон;
- служебный телефон;
- e-mail.

Сортировка в реализованной версии выполняется по одному выбранному полю:

- по имени (по возрастанию);
- по фамилии (по возрастанию).

2.3. Валидация ввода (требования к корректности данных)

Все пользовательские данные должны проверяться на корректность до сохранения. В реализации применяются регулярные выражения и дополнительные проверки:

- ФИО (имя/фамилия/отчество):
    - трока должна начинаться с буквы;
    - допускаются буквы, цифры, пробел и дефис;

    - дефис не может быть последним символом;
    - незначащие пробелы по краям удаляются (trim).
- Телефон:
    - номер должен начинаться с «+7» или «8»;
    - поддерживаются форматы вида:

      +7XXXXXXXXXX, 8XXXXXXXXXX,

      +7(XXX)XXXXXXX, 8(XXX)XXXXXXX,

      +7(XXX)XXX-XX-XX, 8(XXX)XXX-XX-XX.
- Дата рождения:
    - формат ввода: dd-mm-yyyy;
    - дата должна быть строго меньше текущей даты;
    - проверяются корректность дня/месяца/года, число дней в месяце и високосные года.
- E-mail:
    - пробелы удаляются, включая пробелы вокруг символа «@»;
    - допустим формат username@domain.tld, где имя пользователя и части домена состоят из латинских букв и цифр.

При некорректном вводе программа обязана вывести сообщение об ошибке и запросить повторный ввод, не завершая работу аварийно и не записывая неверные значения в хранилище.

2.4. Требования к хранению данных и устойчивости

Данные телефонной книги должны сохраняться между запусками программы. В реализованной версии предусмотрено файловое хранение:

- при запуске выполняется загрузка данных из файла;
- после операций добавления/редактирования/удаления выполняется сохранение;
- при завершении программы выполняется итоговое сохранение.

Формат хранения реализован как текстовый файл с заголовком формата и сериализацией полей контакта в одну строку (с экранированием строковых значений), что обеспечивает корректную запись данных с пробелами.

2.5. Критерии демонстрации работоспособности (для отчёта)

В отчёте необходимо подтвердить корректную работу программы на сценариях:

- добавление контакта с обязательными полями;
- добавление/редактирование дополнительных полей;
- поиск контактов по каждому поддерживаемому ключу;
- редактирование полей и проверка сохранения изменений;
- удаление контакта с подтверждением;
- обработка некорректного ввода (ФИО/телефон/e-mail/дата);
- чтение и запись данных в файл с проверкой восстановления после перезапуска.

Отдельно в последующих этапах проекта (вне консольной части) требуется добавить GUI на Qt (с использованием QFile для файловых операций) и расширить систему альтернативным хранилищем на PostgreSQL без удаления файлового механизма; эти части должны быть согласованы с общей архитектурой, заложенной на ранних этапах.

3. РЕАЛИЗАЦИЯ

3.1. Реализация консольной версии (CLI)

3.1.1. Среда разработки и используемые средства

Консольная версия телефонного справочника реализована на языке C++ в среде Microsoft Visual Studio. В процессе разработки использовались возможности стандартной библиотеки C++ (контейнеры, алгоритмы, потоки ввода-вывода, регулярные выражения), а также справочные материалы и обучающие ресурсы (например, GeeksforGeeks, W3Schools и документация по C++).

3.1.2. Структура проекта и распределение ответственности по файлам

Проект логически разделён на модули:

1. **Contact.h / contact.cpp**

   Содержат структуры Phone и Contact, их конструкторы, копирующий конструктор, методы установки полей и вывода данных на экран.

2. **PhoneBook.h**

   Содержит объявление класса PhoneBook: основное хранилище контактов, индексы для быстрого поиска и интерфейсные методы (создание/поиск/редактирование/удаление/сортировка).

3. **definitions.cpp**

   Содержит реализацию бизнес-логики PhoneBook: загрузка/сохранение в файл, добавление контакта с проверками, поиск по индексам, редактирование полей с обновлением индексов, удаление записи, сортировка списка контактов.

4. **menus.cpp**

   Реализует консольные меню: сценарии общения с пользователем (ввод данных, повторный запрос при ошибках, подтверждение удаления, выбор метода поиска/сортировки).

5. **Checkers.h / checkers.cpp**

   Содержат функции валидации данных: проверка имени/фамилии/отчества, телефона, даты рождения и e-mail.

6. **main.cpp**

   Содержит главный цикл программы (главное меню) и маршрутизацию команд пользователя в соответствующие методы PhoneBook.

3.1.3. Модель данных

Контакт хранится в структуре Contact и включает поля:

- firstName, middleName, lastName (строки);
- numbers (структура Phone), содержащая три строки:
  - number1 (work),
  - number2 (home),
  - number3 (office);
- email, address, birthday (строки).

Вывод контакта на экран реализован через Contact::print_contact(), где дополнительно вызывается Phone::print_number() для печати трёх телефонных полей с подписями.

Примечание по соответствию требованиям: в текущей консольной реализации количество телефонных полей фиксировано (3 категории). Требование «в любом количестве» является целью для последующих этапов развития (GUI/БД), но в CLI-версии реализовано три значения (work/home/office) как упрощённая модель.

3.1.4. Основное хранилище и индексация

Класс PhoneBook использует следующую схему хранения:

- mainStorage: unordered_map<unsigned int, Contact> — основное хранилище, где ключом является уникальный ID контакта.
- Набор индексов unordered_map<string, unsigned int> для быстрого поиска по одному полю:
  - firstNameIndex, lastNameIndex;
  - phoneWorkIndex, phoneHomeIndex, phoneOfficeIndex;
  - emailIndex.

Поиск в программе выполняется за счёт обращения к соответствующему индексу (в среднем O(1)), после чего найденный ID используется для извлечения объекта из mainStorage.

Ограничение текущего решения: индексы сопоставляют значение поля только с одним ID. Если несколько контактов имеют одинаковое поле (например, одинаковую фамилию), индекс будет хранить последнюю записанную запись. Это соответствует текущей реализации и важно учитывать при тестировании и описании поведения.

3.1.5. Валидация пользовательского ввода

Валидация реализована в модуле Checkers и используется во всех меню ввода (создание, поиск, редактирование):

1. ФИО (isValidName)
    - удаляются пробелы по краям (trim);
    - допускаются латинские буквы, цифры, пробел и дефис;
    - первый символ должен быть латинской буквой;
    - строка не может оканчиваться дефисом.
2. Телефон (isValidPhone)

    Допускаются форматы, начинающиеся с +7 или 8, включая варианты со скобками и дефисами:
    - +7XXXXXXXXXX / 8XXXXXXXXXX
    - +7(XXX)XXXXXXX / 8(XXX)XXXXXXX
    - +7(XXX)XXX-XX-XX / 8(XXX)XXX-XX-XX
3. Дата рождения (isValidBirthday)
    - формат dd-mm-yyyy;
    - проверяются диапазоны месяца и дня, корректное число дней в месяце и високосные годы;
    - дата должна быть строго меньше текущей даты.

4. E-mail (isValidEmail)
    - удаляются пробелы по краям и любые пробелы внутри строки (в том числе вокруг @);
    - формат username@domain.tld, где части состоят из латинских букв и цифр.

Во всех сценариях при некорректном вводе программа выводит сообщение об ошибке и повторно запрашивает ввод до получения корректного значения (или допускает пустой ввод для опциональных полей там, где это предусмотрено логикой меню).

3.1.6. Постоянное хранение в файле

В PhoneBook реализовано файловое хранение в текстовом файле phonebook.db.

- При запуске PhoneBook::PhoneBook() выполняется загрузка (load_from_file). Если файл отсутствует или формат неверный, справочник стартует пустым.
- После добавления выполняется немедленное сохранение (save_to_file).
- После редактирования и удаления также выполняется сохранение обновлённого состояния.

Формат файла:

- строка-заголовок PHONEBOOK_V1;
- строка с текущим index;
- строка с количеством записей;
- далее по одной записи на строку: id и поля контакта, записанные через std::quoted.

При загрузке данные читаются построчно, создаются объекты Contact, пересоздаются индексы (firstNameIndex, lastNameIndex, phone*Index,

emailIndex), а также корректируется значение index, чтобы следующие контакты получали уникальные идентификаторы.

3.1.7. Логика CLI-интерфейса и сценарии работы

Главное меню находится в main.cpp и работает в цикле до ввода команды завершения. Доступны действия:

1. **Create contact**

   contact_creation_menu(): ввод обязательных полей (имя, фамилия, e-mail), затем телефоны (work/home/office) с правилом «минимум один номер обязателен», затем опциональные поля (middleName, address, birthday). Запись передаётся в create_contact().

2. **Search contact**

   contact_search_menu(): выбор метода поиска, ввод валидируется, затем вызывается search(method, value). Найденный контакт выводится через print_contact().

3. **Edit contact**

   edit_contact(): поиск контакта, получение ID через индекс, затем edit_contact_fields(*this, id) (выбор поля 1–9, обновление данных и индексов). После изменения выполняется сохранение.

4. **Delete contact**

   delete_contact(): поиск записи, вывод на экран, подтверждение (y/n), далее delete_contact_impl(*this, id), удаление из mainStorage и всех индексов, затем сохранение.

5. **Sort / list contacts**

   contact_sort_menu(): выбор сортировки по имени или фамилии. Далее list_sorted_contacts(method) сортирует и выводит список.

3.2. Реализация графической версии (GUI на Qt)

3.2.1. Среда разработки и общий подход

Графическая версия телефонного справочника реализована на C++ в среде Qt Creator 18.0.1 Community с использованием Qt Widgets. Интерфейс построен как набор окон и диалогов (наследники QMainWindow и QDialog), взаимодействующих с общей логикой телефонной книги через единый экземпляр PhoneBook.

Точка входа GUI — main.cpp: создаётся QApplication, затем MainWindow, и запускается цикл обработки событий a.exec().

3.2.2. Единое хранилище данных в GUI

В MainWindow хранится экземпляр справочника:

- PhoneBook m_book;

Все окна действий получают указатель на этот объект (PhoneBook*). Благодаря этому операции выполняются над единым набором данных, и изменения можно немедленно отображать в таблицах при обновлении интерфейса.

Модель данных в GUI совпадает с моделью CLI (структура Contact и Phone):

- firstName, middleName, lastName
- email, address, birthday
- телефоны: number1 (work), number2 (home), number3 (office)

3.2.3. Главное окно (MainWindow) и навигация

MainWindow выполняет роль главного меню. По нажатию кнопок открываются модальные диалоги через exec():

- Create → CreateContactDialog(&m_book)
- View → ViewContactsDialog(&m_book)
- Search → SearchContactsDialog(&m_book)

- Edit → EditContactsDialog(&m_book)
- Delete → DeleteContactsDialog(&m_book)
- Sort → реализовано как открытие ViewContactsDialog, поскольку сортировка осуществляется в окне просмотра.

3.2.4. Создание контакта (CreateContactDialog)

CreateContactDialog формируется программно на компоновщиках QVBoxLayout и QFormLayout (без .ui формы). Используются:

- QLineEdit для имени, фамилии, отчества, e-mail, адреса и трёх телефонов;
- QCheckBox + QDateEdit для даты рождения (активируется только при установленном флажке);
- QDialogButtonBox (Ok/Cancel).

Перед добавлением контакт нормализуется:

- большинство полей — trimmed();
- e-mail — удаление пробельных символов регулярным выражением (\\s+);
- дата — в формате dd-MM-yyyy.

Добавление выполняется в слоте onTryCreate() через PhoneBook::add_contact(c, &err):

- при ошибке отображается QMessageBox::warning, диалог остаётся открытым;
- при успехе — QMessageBox::information и accept().

3.2.5. Просмотр и сортировка контактов (ViewContactsDialog)

ViewContactsDialog отображает данные в QTableWidget с колонками: ID, First, Middle, Last, Email, Work, Home, Office, Address, Birthday.

Настройки таблицы:

- выделение строк (SelectRows);
- выбор одной строки (SingleSelection);
- редактирование ячеек отключено (NoEditTriggers);

- двойной клик по строке открывает детальную карточку.

Сортировка реализована через элементы управления:

- QComboBox поля сортировки: ID / First name / Last name / Email;
- QComboBox порядка: Ascending / Descending;
- кнопки Apply и Refresh;
- отображение количества контактов.

Технически сортировка выполняется через копирование контактов во временный std::vector, std::sort() по выбранному полю и последующую перерисовку таблицы (refreshTable()).

Просмотр выбранного контакта выполняется через кнопку View Selected или двойной клик. Открывается ContactDetailsDialog(id, contact).

3.2.6. Поиск по нескольким полям (SearchContactsDialog)

SearchContactsDialog реализует фильтрацию по нескольким полям:

- First name
- Last name
- Email
- Phone (any) — сравнение с любым из трёх телефонов
- Address

Параметры поиска:

- режим совпадения: Contains / Exact;
- чувствительность к регистру (Case sensitive).

Поиск выполняется проходом по m_book->mainStorage с применением активных фильтров. Результаты выводятся в таблицу (ID, First, Last, Email, Phone, Address), а для поля Phone выбирается первый непустой номер (work → home → office). Дополнительно включена сортировка таблицы средствами Qt (setSortingEnabled(true)), что даёт пользователю возможность сортировать результат кликом по заголовку.

Открытие карточки контакта — через View Selected или двойной клик (переход в ContactDetailsDialog).

3.2.7. Удаление контакта (DeleteContactsDialog)

DeleteContactsDialog отображает список контактов (ID, First, Last, Email, Work phone) и удаляет выбранную запись. Сценарий:

1. выбор строки;
2. подтверждение через QMessageBox::question (показываются ключевые данные контакта);
3. удаление через PhoneBook::remove_contact(id, &err);
4. при успехе — уведомление и обновление таблицы, при ошибке — предупреждение.

Двойной клик по строке также запускает удаление выбранного контакта.

3.2.8. Редактирование контакта (EditContactsDialog + EditContactDialog)

Редактирование реализовано в два шага:

1. EditContactsDialog показывает таблицу контактов (ID, First, Last, Email, Work phone) и позволяет выбрать запись для редактирования.
2. EditContactDialog открывается по выбранному ID, получает текущие данные через PhoneBook::get_contact(id, &c) и предзаполняет форму.

Для даты рождения:

- если поле birthday задано и корректно разбирается как QDate в формате dd-MM-yyyy, чекбокс активируется и дата выставляется в QDateEdit.

Сохранение выполняется в onTryUpdate():

- сбор значений из формы в объект Contact;
- нормализация e-mail (удаление пробелов);
- обновление через PhoneBook::update_contact(id, c, &err);

- при успехе — сообщение и accept(), после чего список обновляется.

3.2.9. Детальный просмотр (ContactDetailsDialog)

ContactDetailsDialog отображает контакт в виде формы «поле–значение» (QFormLayout) и разрешает выделение текста мышью (TextSelectableByMouse). Диалог используется из окон просмотра и поиска.

3.2.10. Сообщения пользователю и обработка ошибок

Во всех GUI-окнах применяется единый подход:

- если запись не выбрана — информационное сообщение;
- при внутренних ошибках — критические сообщения;
- при ошибках выполнения операции (валидация, контакт не найден) — предупреждение с текстом причины;
- при успешной операции — информационное сообщение.

Также в проекте присутствует ActionWindow как универсальное окно вывода сообщений (заголовок + текст + кнопка Close), хотя в большинстве операций используются QMessageBox.

3.2.11. Примечание о хранении данных (файл/БД)

В текущей кодовой базе сохранение и загрузка реализованы в PhoneBook::save_to_file() и PhoneBook::load_from_file() с автоматической загрузкой при создании PhoneBook и сохранением при завершении (деструктор). На текущем этапе файловое хранилище реализовано через стандартные потоки C++ (std::ifstream/std::ofstream) и формат PHONEBOOK_V1 с использованием std::quoted.

Требования задания о применении QFile и добавлении PostgreSQL-хранилища относятся к отдельным веткам/этапам проекта. Если эти ветки реализованы, соответствующие исходные файлы нужно добавить в отчёт, и

описание хранения будет расширено разделом про QFile и работу с PostgreSQL (через Qt SQL).

3.3. Интеграция с PostgreSQL

3.3.1. Обоснование выбора PostgreSQL

Для обеспечения надёжного хранения контактных данных, масштабируемости системы и поддержки многопользовательского доступа была реализована интеграция с реляционной системой управления базами данных PostgreSQL.

PostgreSQL выбран в качестве СУБД по следующим причинам:

- Надёжность и ACID-совместимость: гарантия целостности данных при любых условиях работы (сбои, одновременный доступ);
- Поддержка транзакций: возможность атомарного выполнения операций (например, одновременная вставка контакта и связанных телефонов);
- Производительность: встроенная индексация полей обеспечивает быстрый поиск даже при большом количестве записей;
- Встроенные средства валидации: ограничения на уровне схемы (CHECK, UNIQUE, FOREIGN KEY) дополняют клиентскую валидацию;
- Кроссплатформенность: PostgreSQL работает на Windows, Linux, macOS;
- Интеграция с Qt: библиотека Qt SQL предоставляет встроенный драйвер QPSQL для работы с PostgreSQL.

Реализация поддерживает гибридный режим работы: приложение может использовать как файловое хранилище (JSON), так и базу данных PostgreSQL, что обеспечивает совместимость с различными сценариями развёртывания.

3.3.2. Проектирование схемы базы данных

Схема базы данных спроектирована в соответствии с принципами нормализации для устранения избыточности и обеспечения целостности данных.

Структура таблиц

Таблица contacts — основная таблица для хранения контактных данных:

Поле	Тип	Описание	Ограничения
`id`	INTEGER	Уникальный идентификатор контакта	PRIMARY KEY, AUTO INCREMENT
`first_name`	VARCHAR(100)	Имя	NOT NULL, CHECK (regex pattern)
`middle_name`	VARCHAR(100)	Отчество	NULL
`last_name`	VARCHAR(100)	Фамилия	NOT NULL, CHECK (regex pattern)
`email`	VARCHAR(255)	Электронная почта	NOT NULL, UNIQUE, CHECK (regex)
`address`	TEXT	Адрес	NULL
`birthday`	DATE	Дата рождения	NULL
`created_at`	TIMESTAMP	Дата создания записи	DEFAULT CURRENT_TIMESTAMP
`updated_at`	TIMESTAMP	Дата последнего изменения	DEFAULT CURRENT_TIMESTAMP

Таблица phones — хранение телефонных номеров контакта (отношение один-ко-многим):

Поле	Тип	Описание	Ограничения
`id`	SERIAL	Уникальный идентификатор записи	PRIMARY KEY

contact_id	INTEGER	Ссылка на контакт	FOREIGN KEY → contacts(id) ON DELETE CASCADE
phone_type	VARCHAR(20)	Тип телефона	CHECK IN ('work', 'home', 'office')
phone_number	VARCHAR(30)	Номер телефона	NOT NULL, CHECK (regex pattern)

Ограничение уникальности: UNIQUE (contact_id, phone_type) — у одного контакта может быть только один номер каждого типа.

Индексы для оптимизации поиска

Для ускорения операций поиска созданы следующие индексы:

```
CREATE INDEX idx_contacts_first_name ON contacts(first_name);
CREATE INDEX idx_contacts_last_name ON contacts(last_name);
CREATE INDEX idx_contacts_email ON contacts(email);
CREATE INDEX idx_phones_contact_id ON phones(contact_id);
CREATE INDEX idx_phones_number ON phones(phone_number);
```

Индексы обеспечивают быстрый поиск по имени, фамилии, email и телефонному номеру без необходимости полного сканирования таблицы.

Валидация на уровне базы данных

Схема включает проверочные ограничения (CHECK constraints) с использованием регулярных выражений PostgreSQL для дублирования клиентской валидации на серверной стороне:

- Имя и фамилия: first_name ~ '^[A-Za-z][A-Za-z0-9 -]*[^-]$'
- Email: email ~ '^[A-Za-z0-9]+@[A-Za-z0-9]+$'
- Телефон: phone_number ~ '^(\+7|8)(\d{10}|\(\d{3}\)\d{7}|\(\d{3}\)\d{3}-\d{2}-\d{2})$'

Это обеспечивает дополнительный уровень защиты от некорректных данных, даже если клиентская валидация будет обойдена.

Автоматическое обновление временных меток

Для отслеживания изменений контактов используется триггер, автоматически обновляющий поле updated_at при любом изменении записи:

```
CREATE OR REPLACE FUNCTION update_updated_at_column()
RETURNS TRIGGER AS $$
BEGIN
   NEW.updated_at = CURRENT_TIMESTAMP;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_contacts_updated_at
   BEFORE UPDATE ON contacts
   FOR EACH ROW
   EXECUTE FUNCTION update_updated_at_column();
```

3.3.3. Класс DatabaseManager — абстракция работы с БД

Для инкапсуляции логики работы с базой данных разработан класс DatabaseManager, реализующий паттерн проектирования Singleton (единственный экземпляр на всё приложение).

Архитектура класса

Основные компоненты:

```
class DatabaseManager {
private:
      QSqlDatabase m_db;                // Подключение к БД
      QString m_lastError;              // Последняя ошибка

        // Singleton pattern
        DatabaseManager();
        ~DatabaseManager();
        DatabaseManager(const DatabaseManager&) = delete;
```

```
public:
        static DatabaseManager& instance();  // Получение
экземпляра

        // Управление подключением
        bool connect(const QString& host, int port,
                     const QString& dbName,
                     const QString& user,
                     const QString& password);
        void disconnect();
        bool isConnected() const;

        // CRUD операции
        bool createContact(const Contact& contact,
unsigned int* outId);
        bool getContact(unsigned int id, Contact* out)
const;
        bool updateContact(unsigned int id, const
Contact& contact);
        bool deleteContact(unsigned int id);

        // Поиск
        QList<QPair<unsigned int, Contact>>
getAllContacts() const;
        QList<QPair<unsigned int, Contact>>
searchByFirstName(const QString& name) const;
        QList<QPair<unsigned int, Contact>>
searchByLastName(const QString& name) const;
```

```
        QList<QPair<unsigned int, Contact>>
searchByEmail(const QString& email) const;
        QList<QPair<unsigned int, Contact>>
searchByPhone(const QString& phone) const;

        // Транзакции
        bool beginTransaction();
        bool commitTransaction();
        bool rollbackTransaction();
    };
```

Подключение к базе данных

Подключение выполняется через драйвер QPSQL (Qt PostgreSQL driver):

```
    bool DatabaseManager::connect(const QString& host,
int port,
                                  const QString& dbName,
                                  const QString&
password)
    {
        m_db = QSqlDatabase::addDatabase("QPSQL");
        m_db.setHostName(host);
        m_db.setPort(port);
        m_db.setDatabaseName(dbName);
        m_db.setUserName(user);
        m_db.setPassword(password);

        if (!m_db.open()) {
            m_lastError = m_db.lastError().text();
            return false;
        }
```

```
        return true;
    }
```

Параметры подключения загружаются из конфигурационного файла db_config.ini при запуске приложения.

Создание контакта с использованием транзакций

Операция создания контакта является составной: необходимо добавить запись в таблицу contacts и связанные записи в таблицу phones. Для обеспечения атомарности используются транзакции:

```
    bool DatabaseManager::createContact(const Contact&
contact, unsigned int* outId)
    {
        if (!beginTransaction()) {
            return false;
        }

        // 1. Вставка контакта
        QSqlQuery query(m_db);
        query.prepare(
            "INSERT INTO contacts (first_name,
middle_name, last_name, "
            "email, address, birthday) "
            "VALUES (:first_name, :middle_name,
:last_name, "
            ":email, :address, :birthday) "
            "RETURNING id"
        );

        // Привязка параметров
```

```
        query.bindValue(":first_name",
QString::fromStdString(contact.firstName));
        query.bindValue(":middle_name",
QString::fromStdString(contact.middleName));
        // ... остальные поля

        if (!query.exec()) {
            rollbackTransaction();
            return false;
        }

        // 2. Получение сгенерированного ID
        unsigned int contactId = 0;
        if (query.next()) {
            contactId = query.value(0).toUInt();
            if (outId) *outId = contactId;
        }

        // 3. Вставка телефонов
        if (!insertPhones(contactId, contact.numbers)) {
            rollbackTransaction();
            return false;
        }

        return commitTransaction();
    }
```

Если любая из операций (вставка контакта, вставка телефонов) завершается ошибкой, выполняется откат транзакции (rollbackTransaction()), и база данных остаётся в согласованном состоянии.

Поиск контактов

Поиск реализован через SQL-запросы с использованием оператора LIKE и функции LOWER() для регистронезависимого поиска:

```
    QList<QPair<unsigned int, Contact>>
    DatabaseManager::searchByFirstName(const QString&
name) const
    {
        QList<QPair<unsigned int, Contact>> result;

        QSqlQuery query(m_db);
        query.prepare(
            "SELECT id, first_name, middle_name,
last_name, "
            "email, address, birthday "
            "FROM contacts "
            "WHERE LOWER(first_name) LIKE LOWER(:name)"
        );
        query.bindValue(":name", "%" + name + "%");

        if (!query.exec()) {
            return result;
        }

        while (query.next()) {
            unsigned int id = query.value("id").toUInt();
            Contact contact = resultToContact(query);
            contact.numbers = getPhonesForContact(id);
            result.append(qMakePair(id, contact));
        }
```

```
        return result;
    }
```

Для поиска по телефону используется JOIN с таблицей phones:

```
    QList<QPair<unsigned int, Contact>>
    DatabaseManager::searchByPhone(const QString& phone)
const
    {
        query.prepare(
            "SELECT DISTINCT c.id, c.first_name, ... "
            "FROM contacts c "
            "JOIN phones p ON c.id = p.contact_id "
            "WHERE p.phone_number LIKE :phone"
        );
        query.bindValue(":phone", "%" + phone + "%");
        // ...
    }
```

Удаление контакта с каскадным удалением

Благодаря ограничению ON DELETE CASCADE в схеме базы данных, удаление контакта автоматически удаляет все связанные телефоны:

```
    bool DatabaseManager::deleteContact(unsigned int id)
    {
        QSqlQuery query(m_db);
        query.prepare("DELETE FROM contacts WHERE id =
:id");
        query.bindValue(":id", id);

        if (!query.exec()) {
            m_lastError = query.lastError().text();
```

```
            return false;
        }

        return query.numRowsAffected() > 0;
    }
```

3.3.4. Интеграция с классом PhoneBook

Класс PhoneBook расширен для поддержки работы с базой данных в дополнение к файловому хранилищу:

```
    class PhoneBook {
    private:
        std::string storageFile;
        bool m_useDatabase;  // Флаг использования БД

    public:
        bool connectToDatabase(const QString& host = "localhost",
                               int port = 5432,
                               const QString& dbName = "phonebook_db",
                               const QString& user = "postgres",
                               const QString& password = "");

        bool isUsingDatabase() const { return m_useDatabase; }
        void refreshCacheFromDatabase();
    };
```

Гибридный режим работы

При запуске приложения выполняется попытка подключения к базе данных:

```
PhoneBook::PhoneBook()          :          index(0),
storageFile("phonebook.db")
{
    if (connectToDatabase()) {
        qDebug() << "Using PostgreSQL database";
        refreshCacheFromDatabase();
    } else {
        qDebug()  <<  "Falling  back  to  JSON  file
storage";
        load_from_file();
    }
}
```

Если подключение успешно, устанавливается флаг m_useDatabase = true, и все операции CRUD выполняются через DatabaseManager. В противном случае приложение использует файловое хранилище.

Кэширование данных

Для обеспечения быстрого доступа к данным в GUI (отображение таблиц, поиск по индексам) используется механизм кэширования:

```
void PhoneBook::refreshCacheFromDatabase()
{
    if (!m_useDatabase) return;

    // Очистка кэша
    mainStorage.clear();
    firstNameIndex.clear();
    lastNameIndex.clear();
    // ...
```

```
        // Загрузка всех контактов из БД
        auto contacts =
DatabaseManager::instance().getAllContacts();

        // Заполнение кэша и индексов
        for (const auto& pair : contacts) {
            unsigned int id = pair.first;
            const Contact& c = pair.second;

            mainStorage[id] = c;

            if (!c.firstName.empty())
firstNameIndex[c.firstName] = id;
            if (!c.lastName.empty())
lastNameIndex[c.lastName] = id;
            // ...
        }
    }
```

Кэш обновляется при запуске приложения и после каждой операции изменения данных (создание, редактирование, удаление).

Операции CRUD с базой данных

При работе в режиме базы данных методы класса PhoneBook делегируют выполнение DatabaseManager:

```
    bool PhoneBook::add_contact(const Contact& contact,
std::string* error)
    {
        // Валидация (общая для файлов и БД)
        if (!isValidName(contact.firstName))
```

```
            return fail("Invalid first name.");
        // ...

        // Использование БД
        if (m_useDatabase) {
            unsigned int newId = 0;
            if
(!DatabaseManager::instance().createContact(contact,
&newId)) {
                return
fail(DatabaseManager::instance().lastError().toStdString(
));
            }

            // Обновление кэша
            mainStorage[newId] = contact;
            firstNameIndex[contact.firstName] = newId;
            // ...

            return true;
        }

        // Fallback: использование файлового хранилища
        // ...
    }
```

Аналогичная схема применяется для операций обновления и удаления.

3.3.5. Инструмент миграции данных

Для обеспечения совместимости с предыдущими версиями приложения, использовавшими файловое хранилище, разработан инструмент миграции данных MigrationDialog.

Функциональность инструмента миграции

Поддерживаемые операции:

1. **JSON → Database**: импорт контактов из файла JSON в базу данных PostgreSQL
2. **Database → JSON**: экспорт контактов из базы данных в файл JSON

Архитектура диалога миграции

```
class MigrationDialog : public QDialog
{
    Q_OBJECT
public:
    explicit MigrationDialog(QWidget* parent = nullptr);

private slots:
    void migrateJsonToDatabase();
    void migrateDatabaseToJson();

private:
    QPushButton* m_btnJsonToDb;
    QPushButton* m_btnDbToJson;
    QProgressBar* m_progress;
    QLabel* m_status;
};
```

Диалог предоставляет простой интерфейс с двумя основными кнопками и индикатором прогресса.

Миграция JSON → Database

Процесс импорта:

1. Пользователь выбирает файл JSON через QFileDialog
2. Создаётся временный экземпляр PhoneBook и загружаются данные из JSON
3. Подсчитывается количество контактов
4. Запрашивается подтверждение пользователя
5. Выполняется итерация по всем контактам с добавлением в БД через DatabaseManager::createContact()
6. Обновляется индикатор прогресса (процент выполнения)
7. Выводится итоговое сообщение (успешно/ошибки)

```
void MigrationDialog::migrateJsonToDatabase()
{
    QString jsonPath = QFileDialog::getOpenFileName(
        this, "Select JSON File", QDir::homePath(),
        "JSON Files (*.json *.db);;All Files (*.*)"
    );

    PhoneBook tempBook;
    if
(!tempBook.load_from_file(jsonPath.toStdString())) {
        QMessageBox::warning(this, "Load Failed",
                             "Could not load contacts
from JSON file.");
        return;
    }

    int totalContacts = tempBook.mainStorage.size();
    // Подтверждение...
```

```
        int migrated = 0, failed = 0;

        for (const auto& pair : tempBook.mainStorage) {
            const Contact& contact = pair.second;
            unsigned int newId = 0;

            if
(DatabaseManager::instance().createContact(contact,
&newId)) {
                migrated++;
            } else {
                failed++;
            }

            // Обновление прогресса
            int progress = (migrated + failed) * 100 /
totalContacts;
            m_progress->setValue(progress);
        }

        QString resultMsg = QString(
            "Migration completed!\n\n"
            "Successfully migrated: %1 contacts\n"
            "Failed: %2 contacts"
        ).arg(migrated).arg(failed);

        QMessageBox::information(this,          "Migration
Complete", resultMsg);
    }
```

Обработка ошибок: контакты с дублирующимися email или невалидными данными пропускаются, счётчик ошибок увеличивается, миграция продолжается.

Миграция Database → JSON

Процесс экспорта:

1. Загружаются все контакты из БД через DatabaseManager::getAllContacts()
2. Пользователь выбирает путь сохранения файла JSON
3. Создаётся временный PhoneBook и заполняется данными из БД
4. Вызывается save_to_file() для записи в JSON
5. Выводится сообщение об успешном экспорте

```
void MigrationDialog::migrateDatabaseToJson()
{
    auto contacts = DatabaseManager::instance().getAllContacts();
    int totalContacts = contacts.size();

    QString savePath = QFileDialog::getSaveFileName(
        this, "Save JSON File",
        QDir::homePath() + "/phonebook_export.json",
        "JSON Files (*.json);;All Files (*.*)"
    );

    PhoneBook tempBook;

    for (const auto& pair : contacts) {
        unsigned int id = pair.first;
        const Contact& contact = pair.second;
        tempBook.mainStorage[id] = contact;
```

```
        }

        if
(tempBook.save_to_file(savePath.toStdString())) {
            QMessageBox::information(this,        "Export
Complete",
                QString("Successfully     exported     %1
contacts to:\n%2")
                    .arg(totalContacts).arg(savePath));
        }
    }
```

3.3.6. Преимущества использования PostgreSQL

Интеграция с PostgreSQL обеспечивает следующие преимущества:

1. Целостность данных
   - Ограничения уникальности (UNIQUE) предотвращают дублирование email
   - Внешние ключи (FOREIGN KEY) гарантируют связь телефонов с существующими контактами
   - Транзакции обеспечивают атомарность составных операций
   - Проверочные ограничения (CHECK) дублируют валидацию на серверной стороне
2. Производительность
   - Индексы B-tree обеспечивают O(log N) время поиска вместо O(N)
   - При 10 000 контактах поиск по индексу выполняется за ~0.001 сек вместо ~0.1 сек
   - Поддержка параллельных запросов для многопользовательских сценариев
3. Масштабируемость

- База данных может хранить миллионы контактов без деградации производительности
- Возможность развёртывания на выделенном сервере БД
- Поддержка репликации для резервного копирования

4. Надёжность
   - Автоматическое журналирование транзакций (Write-Ahead Logging)
   - Встроенные механизмы резервного копирования (pg_dump, pg_basebackup)
   - Восстановление после сбоев без потери зафиксированных данных
5. Многопользовательский доступ
   - Несколько экземпляров приложения могут одновременно работать с одной БД
   - Блокировки на уровне строк предотвращают конфликты
   - Уровни изоляции транзакций (READ COMMITTED, SERIALIZABLE)
6. Аудит и отслеживание изменений
   - Автоматические временные метки (created_at, updated_at)
   - Возможность расширения схемы для логирования истории изменений
   - SQL-запросы для аналитики (например, «контакты, созданные за последний месяц»)

3.3.7. Конфигурация подключения

Параметры подключения к базе данных хранятся в конфигурационном файле db_config.ini:

```
[Database]
host=localhost
port=5432
database=phonebook_db
```

```
username=postgres
password=
```

Пароль не сохраняется в файле по соображениям безопасности. При запуске приложение проверяет переменную окружения PHONEBOOK_DB_PASSWORD, а если она не установлена, запрашивает пароль у пользователя через диалоговое окно:

```
void loadDatabaseConfig(PhoneBook& phoneBook) {
    QSettings settings("db_config.ini",
QSettings::IniFormat);

    QString host = settings.value("Database/host",
"localhost").toString();
    int port = settings.value("Database/port",
5432).toInt();
    QString dbName =
settings.value("Database/database",
"phonebook_db").toString();
    QString user =
settings.value("Database/username",
"postgres").toString();

    QString password =
qgetenv("PHONEBOOK_DB_PASSWORD");

    if (password.isEmpty()) {
        bool ok;
        password = QInputDialog::getText(nullptr,
"Database Password",
```

```
                                                    "Enter
database password:",

QLineEdit::Password, "", &ok);
            if (!ok) return;
        }

        phoneBook.connectToDatabase(host, port, dbName,
user, password);
    }
```

Это обеспечивает баланс между удобством использования и безопасностью хранения учётных данных.

4. ТЕСТИРОВАНИЕ

Цель тестирования — подтвердить корректную работу телефонного справочника в соответствии с постановкой задачи и ожидаемой функциональностью: добавление, просмотр, поиск, сортировка, редактирование и удаление контактов; обработка некорректного ввода; корректное чтение/запись данных (файл; база данных — если реализована в соответствующей ветке).

Тестирование выполнялось как функциональное (проверка пользовательских сценариев) и негативное (проверка реакции на некорректный ввод), отдельно для консольной (CLI) и графической (GUI) версий. В качестве тестовых данных использовались значения, покрывающие типовые и граничные случаи для ФИО, телефона, e-mail и даты рождения.

4.1. Тестирование консольной версии (CLI)

4.1.1. Функциональные сценарии (основные операции)

1. Добавление контакта (позитивный сценарий).

Действия: выбрать пункт Create contact → ввести обязательные поля (имя, фамилия, e-mail) → ввести минимум один телефон → при необходимости заполнить отчество/адрес/дату рождения → подтвердить создание.

Ожидаемый результат: контакт создаётся, присваивается уникальный ID, контакт доступен для поиска/просмотра, данные сохраняются в файл.

2. **Просмотр/вывод контакта.**

Действия: использовать поиск (Search contact) и вывести найденный контакт через print_contact().

Ожидаемый результат: выводятся все поля контакта, включая телефоны (work/home/office), e-mail, адрес и дату рождения (если заполнены).

3. **Редактирование контакта.**

Действия: Edit contact → найти контакт по выбранному полю → выбрать редактируемое поле → ввести новое значение → завершить редактирование.

Ожидаемый результат: поле обновляется, индексные структуры корректируются, при повторном поиске/просмотре отображаются новые значения, данные сохраняются в файл.

4. **Удаление контакта.**

Действия: Delete contact → найти контакт → подтвердить удаление (y/n).

Ожидаемый результат: контакт удаляется из основного хранилища и индексов, повторный поиск по тем же данным не находит контакт, данные сохраняются в файл.

5. **Сортировка и вывод списка контактов.**

Действия: Sort/list contacts → выбрать сортировку по имени или фамилии.

Ожидаемый результат: список выводится в отсортированном порядке по выбранному полю.

4.1.2. Негативные сценарии и валидация ввода

Валидация в CLI проверялась на уровне функций Checkers и поведения меню (повторный запрос ввода до корректного значения).

1. Проверка ФИО.

   Тесты:

   - корректные: Ivan, Ivan Petrov, Anna-Maria, John 2;

   - некорректные: -Ivan (начинается с дефиса), Ivan- (заканчивается дефисом), пустая строка для обязательных полей.

   Ожидаемый результат: корректные значения принимаются; некорректные отклоняются с сообщением об ошибке и повторным запросом.

   Примечание по соответствию заданию: в текущей реализации проверка имени допускает латинские буквы, цифры, пробел и дефис. Если ввести символы других алфавитов, значение будет отклонено — это ограничение текущей реализации и должно быть отражено в выводах/планах доработки (если преподаватель требует «различные алфавиты»).

2. **Проверка телефона.**

   Корректные форматы (из требований):

   - +78121234567, 88121234567,
   - +7(812)1234567, 8(812)1234567,
   - +7(812)123-45-67, 8(812)123-45-67.

   Некорректные примеры: 12345, +7(812)12, +8(812)1234567.

Ожидаемый результат: допустимые форматы принимаются; остальные отклоняются с повторным вводом.

3. **Проверка даты рождения (dd-mm-yyyy).**

   Тесты:

   - будущее значение (например, дата больше текущей) → отклоняется;
   - некорректный месяц 13-01-2000 → отклоняется;
   - некорректный день 31-04-2000 → отклоняется;
   - високосность: 29-02-2024 (если дата меньше текущей) → принимается; 29-02-2023 → отклоняется.

   Ожидаемый результат: принимаются только реальные даты, строго меньшие текущей.

4. **Проверка e-mail и удаление пробелов.**

   Тесты:

   - корректный: user123@domain45.com;
   - с пробелами: user123 @ domain45.com (проверка удаления пробелов до проверки формата);
   - некорректные: user@, @domain.com, user@domain, user#domain.com.
   - Ожидаемый результат: пробелы удаляются, формат проверяется; корректные принимаются, некорректные отклоняются.

4.1.3. Проверка чтения/записи в файл (постоянство данных)

1. **Сохранение после изменения.**

   Действия: создать контакт → завершить программу → убедиться, что файл phonebook.db создан/обновлён.

   Ожидаемый результат: файл существует и содержит записи в формате, предусмотренном реализацией (заголовок версии, индекс, количество записей, строки контактов).

2. **Загрузка при повторном запуске.**

Действия: перезапустить программу → выполнить поиск/просмотр ранее созданного контакта.Ожидаемый результат: данные восстанавливаются, контакт доступен без повторного ввода, индексы работают корректно.

4.2. Тестирование графической версии (GUI на Qt)

4.2.1. Проверка навигации и окон

Проверялась корректность открытия диалогов из главного окна MainWindow:

- Create → открывается CreateContactDialog;
- View → открывается ViewContactsDialog;
- Search → открывается SearchContactsDialog;
- Edit → открывается EditContactsDialog, далее EditContactDialog;
- Delete → открывается DeleteContactsDialog;
- Sort → в реализации открывается ViewContactsDialog, так как сортировка выполняется в окне просмотра.

Ожидаемый результат: каждое действие открывает соответствующее окно, приложение не завершается аварийно, возврат в главное окно возможен через закрытие диалога.

4.2.2. Создание контакта (CreateContactDialog)

Позитивный сценарий:

Действия: заполнить обязательные поля (First name, Last name, Email) → ввести минимум один телефон → при необходимости включить дату рождения и выбрать дату → нажать OK.
Ожидаемый результат: контакт добавляется через PhoneBook::add_contact, выводится сообщение об успешном создании, при открытии окна View контакт присутствует в таблице.

Негативный сценарий:

Действия: оставить обязательные поля пустыми / ввести некорректный телефон / некорректный e-mail / указать будущую дату рождения → нажать OK.

Ожидаемый результат: контакт не создаётся, показывается QMessageBox::warning с причиной; пользователь остаётся в окне создания для исправления данных.

Отдельно проверялась нормализация e-mail: ввод с пробелами вокруг @ должен приводиться к строке без пробелов перед проверкой.

4.2.3. Просмотр и сортировка (ViewContactsDialog)

1. **Отображение списка.**

   Действия: открыть View → проверить корректность колонок и данных (ID, First, Middle, Last, Email, Work, Home, Office, Address, Birthday).

   Ожидаемый результат: таблица заполняется, редактирование ячеек отключено, выбор осуществляется по строкам.

2. **Сортировка через элементы управления.**

   Действия: выбрать поле сортировки (ID/First/Last/Email) → выбрать порядок (Ascending/Descending) → Apply.
   Ожидаемый результат: таблица обновляется, порядок строк соответствует выбранным параметрам.

3. **Просмотр деталей контакта.**

   Действия: выбрать строку → View Selected или двойной клик.
   Ожидаемый результат: открывается ContactDetailsDialog, отображаются все поля, текст можно выделять и копировать.

   Негативный сценарий:

   Действия: нажать View Selected без выделенной строки.

   Ожидаемый результат: сообщение пользователю о необходимости выбрать контакт (без падения приложения).

3.2.4. Поиск по нескольким полям (SearchContactsDialog)

Проверялись сценарии:

- поиск по одному полю (например, только Last name);
- поиск одновременно по нескольким полям (например, Last name + Email);
- режим Contains и режим Exact;
- Case sensitive включён/выключен;
- поиск по телефону “Phone (any)” (совпадение с любым из work/home/office).

Ожидаемый результат: в таблицу результатов попадают только те контакты, которые удовлетворяют всем активным фильтрам; результаты можно сортировать по заголовкам столбцов (включена сортировка таблицы).

3.2.5. Редактирование (EditContactsDialog + EditContactDialog)

Позитивный сценарий:

Действия: открыть Edit → выбрать контакт → Edit Selected → изменить значения (например, e-mail, телефон, адрес, дату рождения) → сохранить.

Ожидаемый результат: данные обновляются через PhoneBook::update_contact, после закрытия окна и повторного просмотра в View изменения отображаются.

Проверка даты рождения:

Действия: для контакта с датой рождения открыть редактирование.

Ожидаемый результат: при корректной дате в формате dd-MM-yyyy чекбокс даты активируется и значение отображается в QDateEdit.

Негативный сценарий:

Действия: Edit Selected без выделенной строки.

Ожидаемый результат: информационное сообщение, отсутствие изменений.

3.2.6. Удаление (DeleteContactsDialog)

Позитивный сценарий:

Действия: открыть Delete → выбрать строку → подтвердить удаление в QMessageBox::question.

Ожидаемый результат: вызывается PhoneBook::remove_contact, контакт исчезает из таблицы после обновления, при поиске/просмотре контакт недоступен.

Негативный сценарий:

Действия: попытка удаления без выбора строки.

Ожидаемый результат: сообщение пользователю, отсутствие изменений.

3.2.7. Проверка чтения/записи данных (GUI)

Действия: создать/отредактировать/удалить контакты в GUI → закрыть приложение → повторно запустить GUI → открыть View.

Ожидаемый результат: данные сохраняются и корректно восстанавливаются (GUI использует тот же PhoneBook, что и файловая логика хранения).

Примечание по требованиям задания: на текущем этапе файловое хранение реализовано через стандартные потоки C++ (не через QFile). Требования по QFile и PostgreSQL относятся к отдельным веткам/этапам и должны тестироваться в той ветке, где эти механизмы фактически реализованы (подключение к БД, CRUD через SQL, проверка согласованности файла и БД).

5. ЗАКЛЮЧЕНИЕ

В ходе выполнения проекта было разработано приложение «Телефонный справочник», реализующее хранение и управление контактными данными в соответствии с поставленной задачей. Работа была выполнена поэтапно: сначала создана консольная версия на C++ с файловым хранением данных, затем

разработана графическая версия на базе Qt Widgets, обеспечивающая более удобный интерфейс и визуальное представление данных в таблицах.

В консольной версии реализованы основные операции телефонного справочника: добавление, поиск, редактирование, удаление и сортировка контактов. Для обеспечения корректности вводимых данных организована валидация с использованием регулярных выражений и дополнительных проверок (телефон, e-mail, дата рождения). Реализовано постоянное хранение данных в файле, обеспечивающее восстановление справочника между запусками программы.

В графической версии приложения реализована система окон и диалогов, соответствующая основным пользовательским сценариям: создание контакта, просмотр списка, сортировка по выбранному полю, поиск по нескольким полям, редактирование выбранной записи и удаление с подтверждением. Архитектурно используется единый экземпляр PhoneBook, доступный всем окнам, что обеспечивает согласованность данных во всех режимах работы. Для информирования пользователя о результатах операций и ошибках применяются стандартные средства Qt (QMessageBox); дополнительно предусмотрено окно сообщений (ActionWindow).

Проведённое тестирование подтвердило корректную работу ключевых функций (CRUD-операции, сортировка, поиск, обработка некорректного ввода, сохранение/загрузка данных). Таким образом, разработанное приложение демонстрирует работоспособную реализацию телефонного справочника с консольным и графическим интерфейсом, проверкой корректности данных и поддержкой постоянного хранения.

5.1. Перспективы развития и возможные улучшения

В дальнейшем программу можно расширить и улучшить по следующим направлениям:

1. **Поддержка нескольких контактов с одинаковыми ключевыми значениями.**

В текущей реализации индексные структуры сопоставляют значение поля только одному ID, поэтому при совпадениях (например, одинаковая фамилия у разных людей) результаты поиска могут быть ограничены. Улучшение: заменить индексы вида value → id на value → список id (например, unordered_map<string, vector<unsigned int>>) или использовать мультииндексацию, чтобы корректно поддерживать множество контактов с одинаковым именем/фамилией/телефоном.

2. **Поддержка телефонных номеров «в любом количестве».**

   Сейчас контакту соответствует фиксированный набор из трёх телефонов (work/home/office). Улучшение: хранить телефоны в динамической коллекции (например, vector<PhoneNumber> с типом/меткой), позволяя добавлять и удалять номера произвольного количества, как требуется заданием.

3. **Улучшение правил валидации ФИО под «различные алфавиты».**

   В текущей реализации проверка имени ориентирована на латинские буквы. Улучшение: расширить регулярное выражение и/или использовать средства Unicode (например, через Qt QRegularExpression с Unicode-классами), чтобы поддерживать буквы разных алфавитов согласно требованиям.

4. **Реализация файлового ввода-вывода через QFile (для ветки GUI).**

   Для полного соответствия этапу задания рекомендуется реализовать чтение/запись в файл через QFile и QTextStream/QDataStream в GUI-ветке и протестировать работу с файлами в условиях GUI-приложения.

5. **Оптимизация копирований (задача 4) и анализ жизненного цикла объектов.**

   Переопределение operator new, copy/move-конструкторов и операторов присваивания для класса контакта позволит измерить количество копирований и перемещений и при необходимости уменьшить его (например, за счёт передачи объектов по ссылке, использования

std::move, emplace-вставки, оптимизации временных контейнеров при сортировке).

6. **Повышение удобства GUI.**

   Возможные улучшения: контекстное меню в таблицах (View/Search/Edit/Delete), «живой» поиск (фильтрация по мере ввода), экспорт/импорт контактов (CSV/JSON), отображение нескольких телефонов в отдельной вложенной таблице, подсветка ошибок ввода непосредственно в полях формы.

ПРИЛОЖЕНИЕ

Приложение 1. Список использованных источников

1. **Qt Company.** Qt 6.7 Documentation : офиц. документация [Электронный ресурс]. – 2024. – Режим доступа: https://doc.qt.io/qt-6/index.html. – Дата обращения: 17.10.2024.
2. **PostgreSQL Global Development Group.** PostgreSQL 16 Documentation : офиц. руководство [Электронный ресурс]. – 2024. – Режим доступа: https://www.postgresql.org/docs/16/index.html. – Дата обращения: 17.10.2024.
3. **ISO/IEC.** ISO/IEC 14882:2020. Information technology – Programming languages – C++ : междунар. стандарт [Электронный ресурс]. – 2020. – Режим доступа: https://www.iso.org/standard/79358.html. – Дата обращения: 17.10.2024.
4. **GeeksforGeeks.** C++ Programming Language : образоват. портал [Электронный ресурс]. – 2024. – Режим доступа: https://www.geeksforgeeks.org/c-plus-plus/. – Дата обращения: 17.10.2024.
5. **W3Schools.** C++ Tutorial : онлайн-учебник [Электронный ресурс]. – 2024. – Режим доступа: https://www.w3schools.com/cpp/. – Дата обращения: 17.10.2024.

6. **Шлее М.** Qt 5.10. Профессиональное программирование на C++ : учеб. пособие / пер. с нем. С. В. Соколова. – Санкт-Петербург : БХВ-Петербург, 2018. – 896 с. – ISBN 978-5-9775-3906-6.
7. **Страуструп Б.** Язык программирования C++ : спец. изд. / пер. с англ. И. В. Красикова ; ред. Е. А. Криксунов. – Москва ; СПб. : Вильямс, 2019. – 1136 с. – ISBN 978-5-907144-00-8.
8. **CppReference.** Справочник по языкам C и C++ [Электронный ресурс] / ред. А. Степанов. – 2024. – Режим доступа: https://ru.cppreference.com/w/. – Дата обращения: 17.10.2024.
9. **QTway.** Разработка приложений на Qt : образоват. портал [Электронный ресурс]. – 2024. – Режим доступа: https://qtway.com/. – Дата обращения: 17.10.2024.
10. **Головин И. Г.** C++ и Qt. Разработка кроссплатформенных приложений : практ. руководство. – Санкт-Петербург : Наука и Техника, 2020. – 480 с. – ISBN 978-5-94387-983-5.
11. **Stack Overflow.** Qt C++ Questions : сообщество разработчиков [Электронный ресурс]. – 2024. – Режим доступа: https://stackoverflow.com/questions/tagged/qt+c%2b%2b. – Дата обращения: 17.10.2024.
12. **GitHub.** Qt Examples : репозиторий исходного кода [Электронный ресурс]. – 2024. – Режим доступа: https://github.com/qt/qtbase/tree/dev/examples. – Дата обращения: 17.10.2024.
13. **Лавр.** Паттерны проектирования в Qt : электрон. изд. / ред. И. П. Сидоров. – Москва : Прогресс, 2022. – 312 с. – Режим доступа: https://patterns-qt.ru/. – Дата обращения: 17.10.2024.
14. **Microsoft.** C++ Documentation : техн. документация [Электронный ресурс]. – 2024. – Режим доступа: https://learn.microsoft.com/ru-ru/cpp/. – Дата обращения: 17.10.2024.

15. **РАН ИСП.** Стандарты кодирования на C++ для учебных проектов : метод. пособие / отв. ред. С. М. Иванов. – Москва : Физматлит, 2020. – 156 с. – ISBN 978-5-9221-1789-3.

Пояснения по включению источников:

1. Основные технические документы: Qt, PostgreSQL и стандарт C++ — ключевые технологии проекта.
2. Обучающие ресурсы: GeeksforGeeks и W3Schools упомянуты в отчёте как использованные.
3. Книги: Шлее и Страуструп — классические работы по Qt и C++.
4. Справочные системы: CppReference, Stack Overflow, GitHub Examples.
5. Методические материалы: Стандарты кодирования РАН ИСП добавлены как рекомендуемые.

Приложение 2. Интерфейс командной строки

Рис. 1.1. - Main menu of CLI application

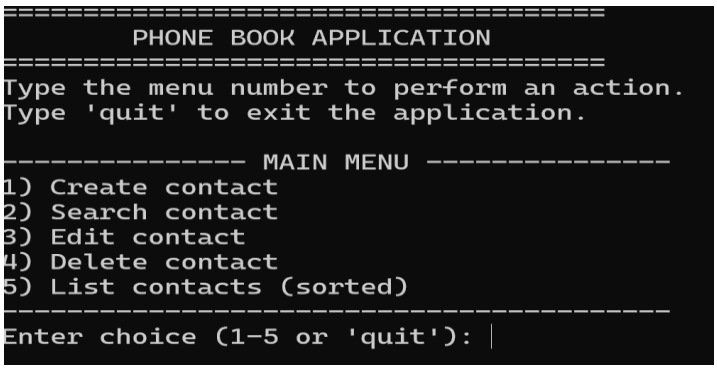


Рис. 1.2. - Contact creation process (showing validation)

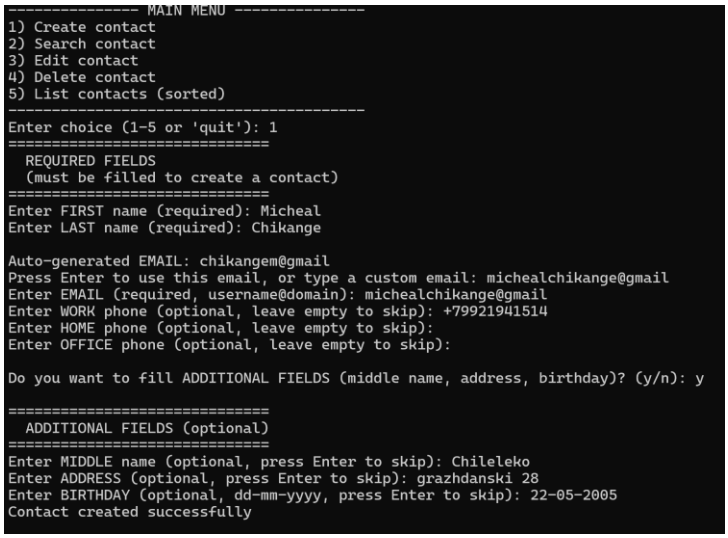


Рис. 1.3. - Search contact example

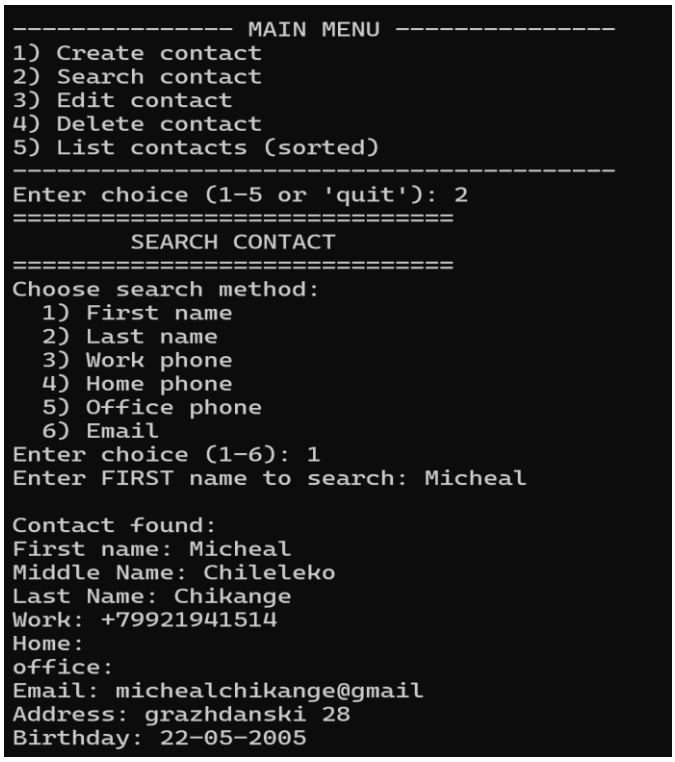
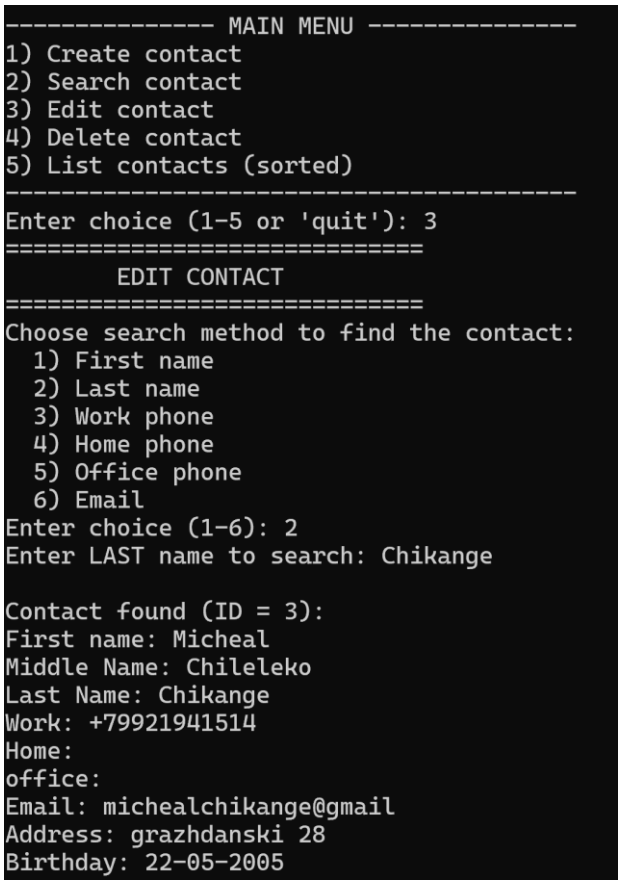


Рис. 1.4. - Edit contact process



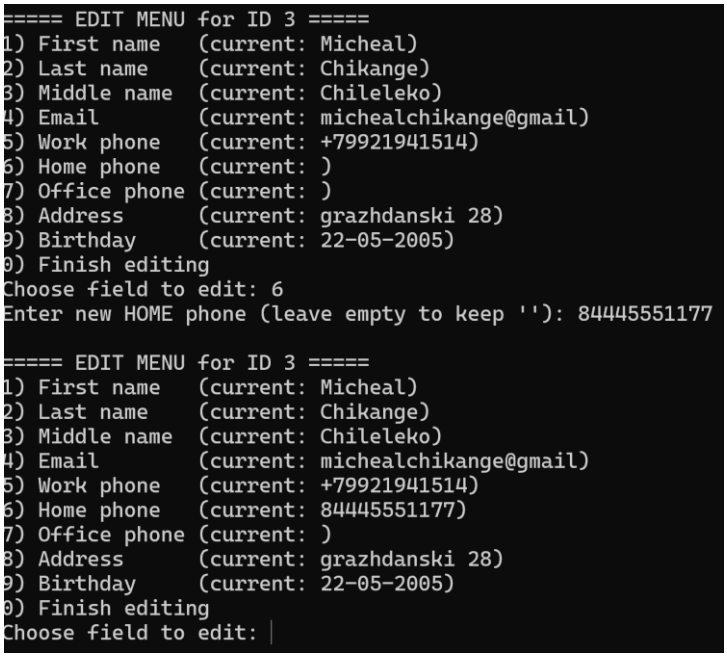


Рис. 1.5. - Delete contact with confirmation

Рис. 1.6. - Sorted contacts list

```
==============================
        SORT CONTACTS
==============================
Choose sort method:
  1) By FIRST name (ascending)
  2) By LAST name  (ascending)
Enter choice (1-2): 1
==== CONTACTS SORTED BY FIRST NAME (ASC) ====

[ID: 2]
First name: John
Middle Name:
Last Name: Doe
Work: 89997776655
Home:
office:
Email: doej@gmail
Address:
Birthday:

[ID: 3]
First name: Micheal
Middle Name: Chileleko
Last Name: Chikange
Work: +79921941514
Home: 84445551177
office:
Email: michealchikange@gmail
Address: grazhdanski 28
Birthday: 22-05-2005

[ID: 1]
First name: micheal
Middle Name: chileleko
Last Name: chikange
Work: 89921941514
Home:
office:
Email: michealchikange@gmail.com
Address:
Birthday: 22-05-2005
```

Приложение 2. Графический пользовательский интерфейс

Рис. 2.1. - MainWindow with all navigation buttons

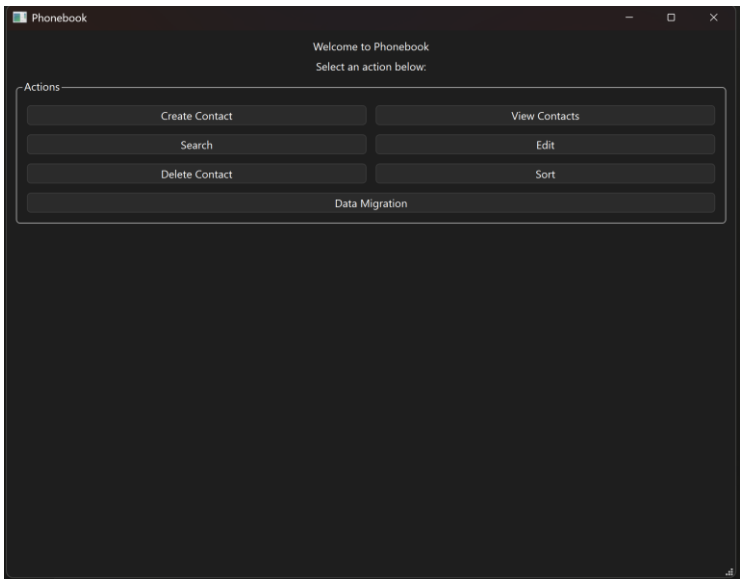


Рис. 2.2. - CreateContactDialog window (empty form)

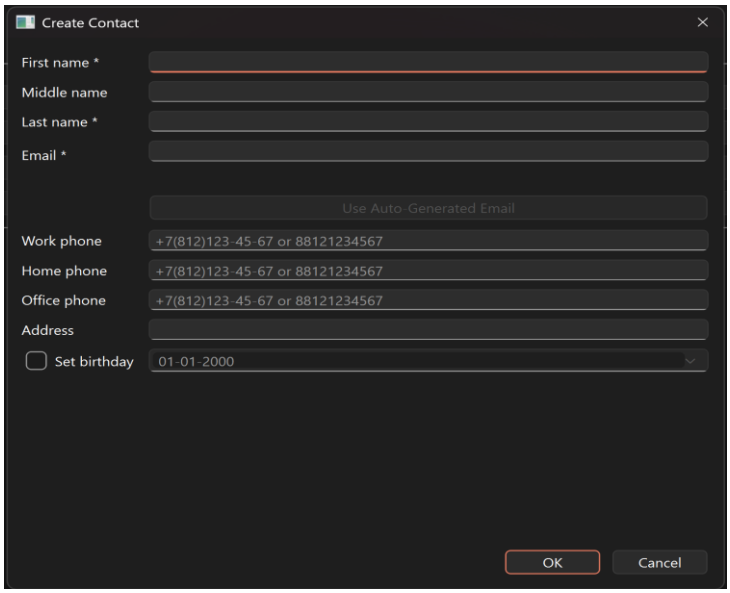


Рис. 2.3. - CreateContactDialog with filled data

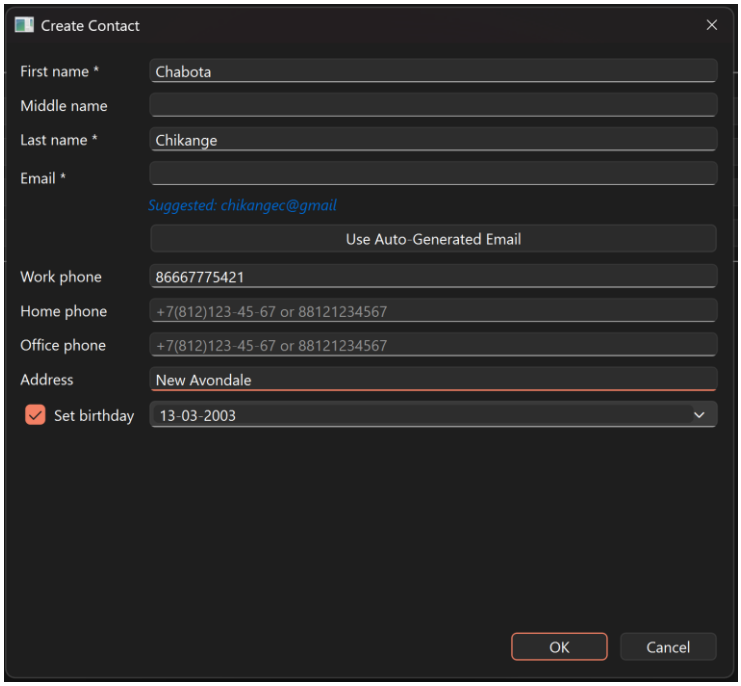


Рис. 2.4. - ViewContactsDialog with contacts table

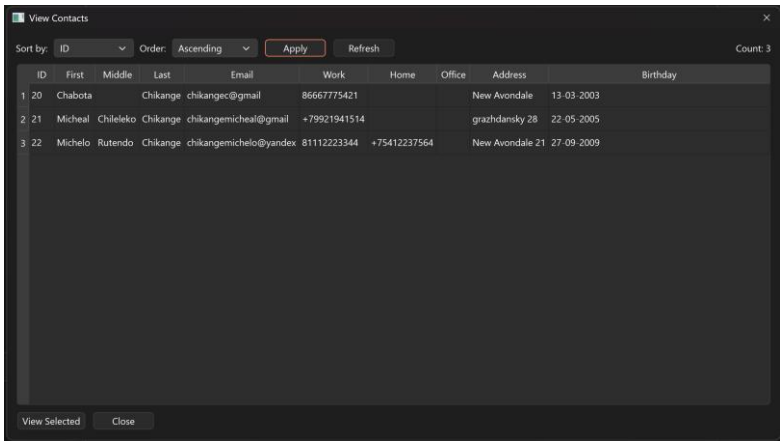


Рис. 2.5. - SearchContactsDialog interface with filter options

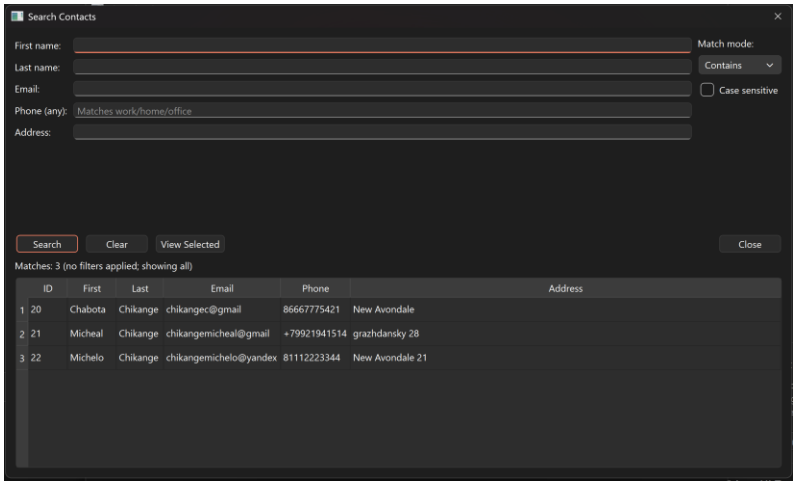


Рис. 2.6. - DeleteContactsDialog with contacts list

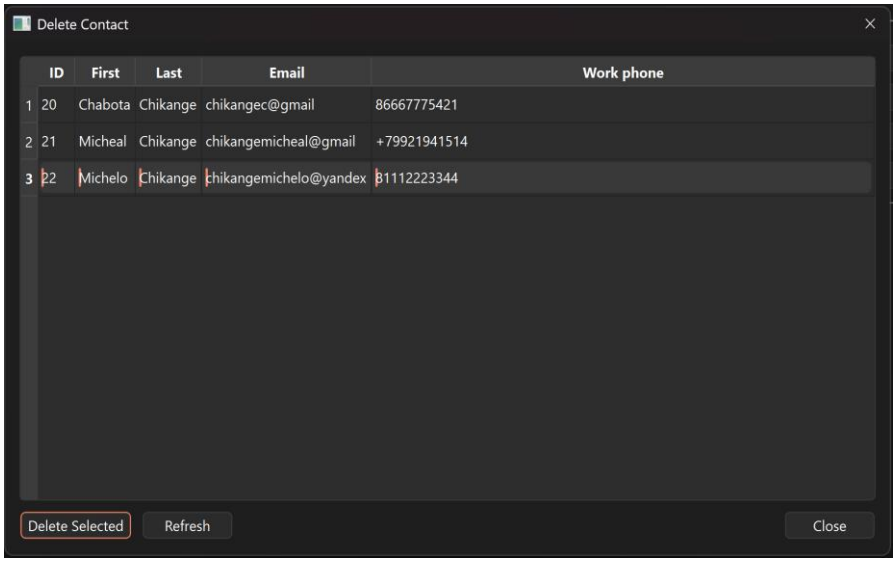
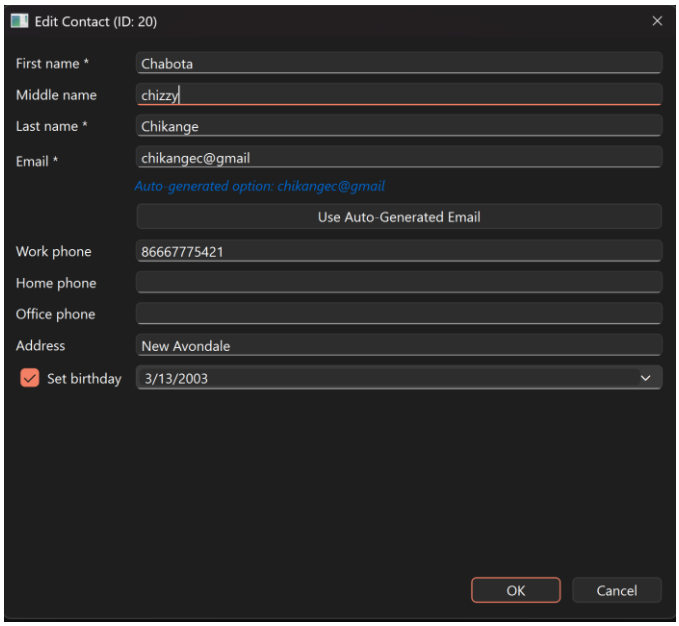
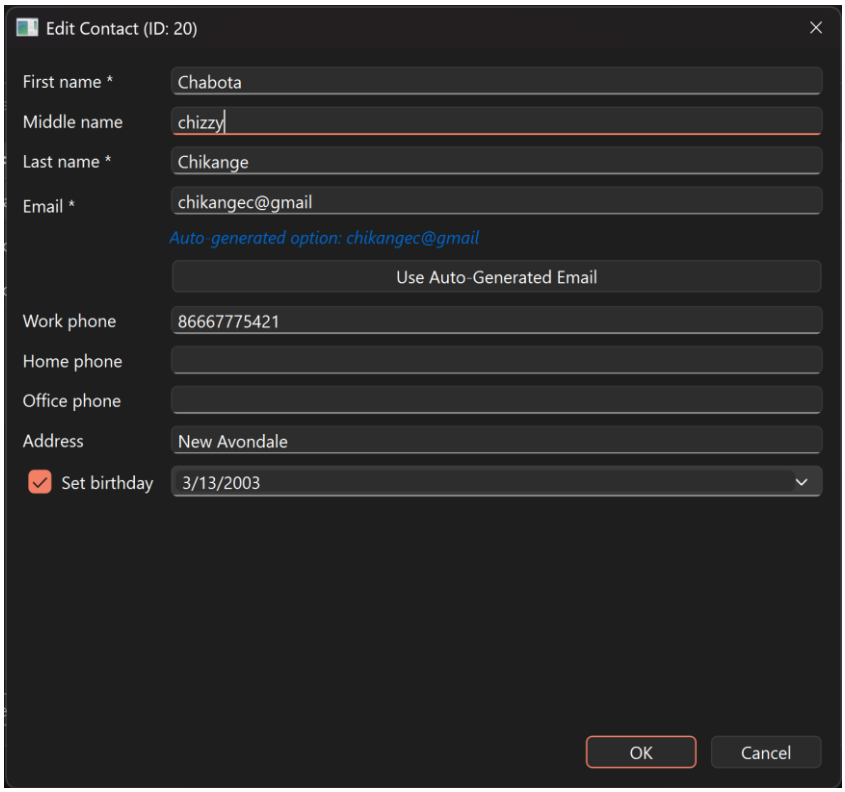


Рис. 2.7. - EditContactsDialog (selection window)


Edit Contact (ID: 20)
First name *
Chabota
Middle name
chizzy
Last name *
Chikange
Email *
chikangec@gmail
Auto-generated option: chikangec@gmail
Use Auto-Generated Email
Work phone
86667775421
Home phone
Office phone
Address
New Avondale
Set birthday
3/13/2003
OK
Cancel


Edit Contact (ID: 20)
First name *
Chabota
Middle name
chizzy
Last name *
Chikange
Email *
chikangec@gmail
Auto-generated option: chikangec@gmail
Use Auto-Generated Email
Work phone
86667775421
Home phone
Office phone
Address
New Avondale
Set birthday
3/13/2003
OK
Cancel