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

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление: 02.03.01 «Математика и компьютерные науки»

Отчет о выполнении курсовой работы

Объектно-ориентированное программирование
Создание телефонного справочника с использованием библиотеки Qt

Студент,
группа 5130201/30002 _____ Путята М. А.

Преподаватель _____ Кирпиченко С. Р.

«_____»_____ 2024 г.

Санкт-Петербург
2024

Содержание

Введение **4**

1 Постановка задачи **5**

2 Реализация **6**

2.1 Класс Page 6

2.1.1 Конструктор 7

2.1.2 Методы для получения значений переменных в классе Page (геттеры) 7

2.1.3 Метод getAll_Data_Strings 8

2.1.4 Метод getPhones_Layout 8

2.1.5 Методы для задания значений переменных в классе Page (сеттеры) 9

2.1.6 Метод add_row_to_numbers 9

2.1.7 Метод check_line 10

2.1.8 Метод getPhone_types_str 10

2.2 Класс Book 11

2.2.1 Конструктор 11

2.2.2 Метод add_Page 12

2.2.3 Методы для получения значений переменных (геттеры) 12

2.2.4 слот add_Row 13

2.2.5 слот check_item 13

2.2.6 Слоты update и clear 15

2.2.7 Слот delete_row 16

2.2.8 Слот confirmExit 16

2.3 Функции 17

2.3.1 Функция writeBookToJsonFile 17

2.3.2 Функция readBookFromJsonFile 18

2.3.3 Функции для выключения редактирования телефонной книги . 19

2.3.4 Функция search 20

2.4 Хранение константных значений 22

3 Тестирование приложения **24**

Заключение **31**

Приложение А. Исходный код класса Page **32**

А.1 Исходный код файла Page.h 32
А.2 Исходный код файла Page.cpp 34

Приложение Б. Исходный код класса Book **37**
Б.2 Исходный код файла Book.h 37
Б.2 Исходный код файла Book.cpp 38

Приложение В. Исходный код функции main **47**

Введение

В современном мире хранение и обработка данных является неотъемлемой частью эффективного управления информацией. В этом контексте телефонные справочники играют важную роль, позволяя пользователям удобно организовывать и хранить контактную информацию, такую как имена, адреса и номера телефонов. Чтобы создать такой инструмент, необходимо использовать надежные технологии, способствующие быстрому и эффективному доступу к данным.

В данной работе будет использоваться библиотека Qt, позволяющая создавать кроссплатформенные приложения с графическим интерфейсом. Qt предлагает мощные инструменты для разработки, такие как Qt Widgets для создания интерфейсов и класс QFile для работы с файлами, что делает его идеальным выбором для разработки телефонного справочника.

1 Постановка задачи

Необходимо написать приложение телефонного справочника с использованием библиотеки Qt, которое будет обеспечивать удобное управление контактными данными. В справочник должен содержать следующие поля для хранение информации: имя, фамилия, отчество, адрес, дата рождения, Email и телефонные номера (рабочий, домашний, служебный) с возможностью добавления нескольких номеров.

Валидация данных:

1. Фамилия, имя и отчество:
   - Должны содержать только буквы, цифры, дефисы и пробелы.
   - Начинаются с заглавной буквы.
   - Не могут начинаться или оканчиваться на дефис.
   - Все незначащие пробелы должны удаляться.
2. Дата рождения:
   - Должна быть раньше текущей даты.
   - Месяц может быть от 1 до 12, а день от 1 до 31 с учётом разного количества дней в месяцах и високосных годов.
3. Email:
   - Должен состоять из имени пользователя (латиница и цифры), символа «@» и домена (латиница и цифры).
   - Все незначащие пробелы должны удаляться.
4. Телефон:
   - Должен быть в международном формате. Например, +7 812 1234567, 8(800)123-1212.
   - Должен храниться в виде последовательности цифр.

В приложении должны быть доступны следующие операции над записями: добавление новых записей, удаление существующих, редактирование любого поля, сортировка данных и поиск по выбранному полю. Информации телефонного справочника необходимо хранить в файле, для чтения и записи данных использовать класс QFile. Для обеспечения удобного отображения данных пользователю использовать таблицы (класс QTableWidget). Проверку пользовательского ввода следует организовать с использование регулярных выражений.

Также в задание были добавлены дополнительные требования: реализовать добавление данных из файла, сделать возможным отключение редактирования данных, сделать возможным удаление сразу нескольких строчек.

2 Реализация

В этом разделе будет описана структура созданного приложения по классам.

2.1 Класс Page

Объекты класса Page можно сказать, что являются минимальными элементами всего телефонного справочника. В экземпляра этого класса содержится вся информация об одном человеке: Фамилия, имя, отчество, адрес, дата рождения, Email и номера телефонов. Для хранения данных используются классы из библиотеки Qt: QString - для фамилии, имени, отчества, адреса и Email'а, QDate - для хранения даты рождения, QVector - для хранения номеров телефонов. Причём номера телефонов хранятся в двух объектах: QVector<QString> - для хранения самих номеров телефонов и QVector<int> - для хранения типа номера телефона (домашний, рабочий или личный).

Класс Page унаследован от класса QWidget из библиотеки Qt для обеспечения возможности использования механики сигналов и слотов, а также более удобной реализации отображения диалоговых окон для предупреждения об ошибках ввода или информирования пользователя. Сигнатура класса Page представлена на листинге 1.

```
1  class Page : public QWidget {
2    Q_OBJECT;
3  private:
4    QString Surname;
5    QString Name;
6    QString Patronymic;
7    QString Address;
8    QDate DOB;
9    QString Email;
10   QVector<QString> Phone_Numbers;
11   QVector<int> Phone_types;
12 public:
13   Page();
14   Page(QString surname_, QString name_, QString patronymic_, QString
       address_, QDate dob_, QString email_, QVector<QString> phone_numbers_
       , QVector<int> phone_types_);
15   QString getSurname() const;
16   QString getName() const;
17   QString getPatronymic() const;
18   QString getAddress() const;
19   QDate getDOB() const;
20   QString getEmail() const;
21   QVector<QString> getPhone_Numbers() const;
22   QVector<int> getPhone_types();
23   QVector<QString> getPhone_types_str();
24   QStringList getAll_Data_Strings() const;
25   QGridLayout* getPhones_Layout();
26   void setSurname(QString surname_);
27   void setName(QString name_);
28   void setPatronymic(QString patronymic_);
29   void setAddress(QString address_);
30   void setDOB(QDate dob_);
```

```
31   void setEmail(QString email_);
32   void setPhone_Numbers(QVector<QString> phone_numbers_);
33   void setPhone_types(QVector<int> new_phone_types);
34   void check_line(QLineEdit* line, int n);
35   void check_combo(int n, int index);
36 signals:
37   void numbers_updated();
38 public slots:
39   void add_row_to_numbers();
```

Листинг 1. Сигнатура класса Page

2.1.1 Конструктор

Конструктор инициализирует переменные для хранения информации об одном человеке. Код представлен на листинге 2.

```
1  Page::Page(QString surname_, QString name_, QString patronymic_, QString
       address_, QDate dob_, QString email_, QVector<QString>
     phone_numbers_, QVector<int> phone_types_) {
2    Surname = surname_;
3    Name = name_;
4    Patronymic = patronymic_;
5    Address = address_;
6    DOB = dob_;
7    Email = email_;
8    Phone_Numbers = phone_numbers_;
9    Phone_types = phone_types_;
10 }
```

Листинг 2. Конструктор Page

2.1.2 Методы для получения значений переменных в классе Page (геттеры)

К данном классе определены методы для получения значения любой переменной: getSurname, getName, getPatronymic, getAddress, getDOB, getEmail, getPhone_Numbers, getPhone_types. Коды этих методов предствалены на листинге 3.

```
1  QString Page::getSurname() const { return Surname; }
2  QString Page::getName() const { return Name; }
3  QString Page::getPatronymic() const { return Patronymic; }
4  QString Page::getAddress() const { return Address; }
5  QDate Page::getDOB() const { return DOB; }
6  QString Page::getEmail() const { return Email; }
7  QVector<QString> Page::getPhone_Numbers() const { return Phone_Numbers;
     }
8  QVector<int> Page::getPhone_types() { return Phone_types; }
```

Листинг 3. Коды методов для получения значений переменных

2.1.3 Метод getAll_Data_Strings

Данный метод создан для более удобного более удобного добавления страницы в телефоннкю книгу. Метод getAll_Data_Strings возвращает QStringList, который содержит в себе значения всех переменных в классе Page, переведённые в тип данных QString. Код метода представлен на листинге 4.

```
QStringList Page::getAll_Data_Strings() const{
  QStringList Out;
  Out << (Surname.isEmpty() ? Lvalues[static_cast<int>(Values::null)] :
   Surname);
  Out << (Name.isEmpty() ? Lvalues[static_cast<int>(Values::null)] :
   Name);
  Out << (Patronymic.isEmpty() ? Lvalues[static_cast<int>(Values::null)]
    : Patronymic);
  Out << (Address.isEmpty() ? Lvalues[static_cast<int>(Values::null)] :
   Address);
  Out << (DOB.isNull() ? Lvalues[static_cast<int>(Values::null)] : DOB.
   toString("dd.MM.yyyy"));
  Out << (Email.isEmpty() ? Lvalues[static_cast<int>(Values::null)] :
   Email);
  return Out;
}
```

Листинг 4. Код метода getAll_Data_Strings

2.1.4 Метод getPhones_Layout

Данный метод создан для более удобного отображения номеров пользователю. Метод getPhones_Layout создаёт QGridLayout, который содержит в себе кнопку QPushButton "Добавить номер текстовое поле QLineEdit для отображения и ввода номера телефона и комбинированное поле со списком QComboBox для выбора типа номера телефона (домашний, рабочий или личный). Причём при нажатии на кнопку "Добавить номер"в QGridLayout добавляется ещё одно поле для ввода номера телефона и комбинированное поле со списком возможных типов номера. Код метода getPhones_Layout представлен на листинге 5.

```
QGridLayout* Page::getPhones_Layout() {
  QGridLayout* layout_phones = new QGridLayout();
  QStringList str_types = { QStringLiteral("Домашний"), QStringLiteral("
   Рабочий") , QStringLiteral("Личный") };
  for (int i = 0; i < Phone_Numbers.size(); i++) {

    QLineEdit* line = new QLineEdit();
    QComboBox* types = new QComboBox();

    line->setText(Phone_Numbers[i]);
    types->addItems(str_types);
    types->setCurrentIndex(Phone_types[i]);

    layout_phones->addWidget(line, i, 0);
    QObject::connect(line, &QLineEdit::editingFinished, [line, i, this
   ]() {
      check_line(line, i);
      });
    layout_phones->addWidget(types, i, 1);
```

```
18      QObject::connect(types, QOverload<int>::of(&QComboBox::
        currentIndexChanged), [i, this](int index) {
19          check_combo(i, index);
20          });
21    }
22    QPushButton* add_number_button = new QPushButton(QStringLiteral("
        Добавить номер"));
23    layout_phones->addWidget(add_number_button);
24    QObject::connect(add_number_button, &QPushButton::clicked, this, &Page
        ::add_row_to_numbers);
25    return layout_phones;
26  }
```

Листинг 5. Код метода getPhones_Layout

2.1.5 Методы для задания значений переменных в классе Page (сеттеры)

В классе определены следующие методы для задания значений переменных: setSurname, setName, setPatronymic, setAddress, setDOB, setEmail, setPhone_Numbers и setPhone_types. Коды этих методов представлены на листинге 6.

```
1  void Page::setSurname(QString surname_) { Surname = surname_; }
2  void Page::setName(QString name_) { Name = name_; }
3  void Page::setPatronymic(QString patronymic_) { Patronymic = patronymic_
       ; }
4  void Page::setAddress(QString address_) { Address = address_; }
5  void Page::setDOB(QDate dob_) { DOB = dob_; }
6  void Page::setEmail(QString email_) { Email = email_; }
7  void Page::setPhone_Numbers(QVector<QString> phone_numbers_) {
       Phone_Numbers = phone_numbers_; }
8  void Page::setPhone_types(QVector<int> new_phone_types) { Phone_types =
       new_phone_types; }
```

Листинг 6. Коды методов для задания значений переменных

2.1.6 Метод add_row_to_numbers

Метод add_row_to_numbers выполняет добавление информации о новом номере в переменные, отвечающие за хранение номеров телефонов (QVector<QString> Phone_Numbers, QVector<int> Phone_types). В конце выпонения метода посылается сигнал numbers_updated, при появлении которого обновляется телефонная книга. Код метода представлен на листинге 7.

```
1  void Page::add_row_to_numbers() {
2    Phone_Numbers.push_back(Lvalues[static_cast<int>(Values::null)]);
3    Phone_types.push_back(0);
4    emit numbers_updated();
5  }
```

Листинг 7. Код метода add_row_to_numbers

2.1.7 Метод check_line

Этот метод обрабатывает пользовательский ввод номера телефона с помощью регулярного выражения и по возможности приводит его к общему формату и убирает лишние и добавляет новый номер в вектор Phone_Numbers. При неправильном наборе номера, метод создаёт диалоговое окно QMessageBox, для предупреждения пользователя о неправильно введённом номере. Код метода представлен на листинге 7.

```
void Page::check_line(QLineEdit* line, int n) {
  QString input = line->text().trimmed();
  if (line->text() == QString(Lvalues[static_cast<int>(Values::null)]))
   {
    return;
  }
  QRegExp regex("\\+?[ ]?[0-9][0-9]?[0-9]?[ ]?(\\(([0-9]{3}\\))
   |([0-9]{3}))[ ]?[0-9]{3}(([0-9]{4})|([-][0-9]{2}[ -]?[0-9]{2})|(
   [0-9]{2} [0-9]{2}))");
  bool isValid = regex.exactMatch(input);
  if (not isValid and line->text() != QString(Lvalues[static_cast<int>(
   Values::null)])) {
    line->setText(Lvalues[static_cast<int>(Values::null)]);
    Phone_Numbers[n] = Lvalues[static_cast<int>(Values::null)];
    QMessageBox::warning(this, Lerrors[static_cast<int>(Errors::
   incorrect_data)], Lerrors[static_cast<int>(Errors::incorrect_phone)])
   ;
  }
  else {
    QString number_removed = line->text().remove("+").remove(" ").remove
   ("-").remove("(").remove(")");

    QString lasts = number_removed.right(10);
    QString firsts = number_removed.left(number_removed.size() - 10);
    QString out = firsts + " (" + lasts.left(3) + ") " + lasts.mid(3, 3)
    + "-" + lasts.mid(6, 2) + "-" + lasts.mid(8, 2);
    if (line->text().startsWith("+")) {
      out = "+" + out;
    }
    line->setText(out);
    Phone_Numbers[n] = out;
  }
}
```

Листинг 8. Код метода check_line

2.1.8 Метод getPhone_types_str

Метод getPhone_types_str создан для получения типов номеров телефонов в виде QString. Код метода представлен на листинге 9.

```
QVector<QString> Page::getPhone_types_str() {
  QVector<QString> out;
  for (auto i : Phone_types) {
    out.push_back(QString::number(i));
  }
  return out;
```

```
}
```

Листинг 9. Код метода getPhone_types_str

2.2 Класс Book

Класс Book создан для хранении информации не об одном человеке (как класс Page), а уже о всех людях записанных в телефонную книгу. Класс имеет слеующие поля: QVector<Page*> Pages - вектор для хранения указателей на страницы определённых людей, QTableWidget* Table - указатель на таблицу, которая будет отображаться пользователю и с помощью которой он сможет заносить данные в книгу и удалять их, int width - переменная, которая хранит ширину таблицы для правильного отображения пользователю.

Данный класс унаследован от класса QObject из библиотеки Qt для возможности использования слотов. Сигнатура класса Book представлена на листинге 10.

```
class Book : public QObject{
  Q_OBJECT;
private:
  QVector<Page*> Pages;
  QTableWidget* Table;
  int width;
public:
  Book(QWidget* parent_);
  void add_Page(Page* new_page, bool Flag = true);

  QTableWidget* get_Table();
  int get_Width();
  QVector<Page*> getPages();
  void clear();

public slots:
  void add_Row();
  void check_item(QTableWidgetItem* item);
  void update();
  void delete_row();
  void confirmExit(QApplication* app, QString Name);
};
```

Листинг 10. Сигнатура класса Book

2.2.1 Конструктор

В конструкторе класса Book создаётся таблица QTableWidget для отображения телефонной книги и задаётся общая ширина таблицы width. Также в конструкторе связывается сигнал об внесении изменений пользователем в таблицу со слотом класса Book check_item, который будет рассмотрен позже. Код конструктора класса Book представлен на листинге 11.

```
Book::Book(QWidget* parent_) {
  Table = new QTableWidget(0, Lheaders.size(), parent_);
  Table->setHorizontalHeaderLabels(Lheaders);
```

```
  Table->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
  Table->resizeColumnsToContents();
  width = 0;
  for (int i = 0; i < Lheaders.size(); i++) {
    if (Table->columnWidth(i) < width_column) {
      Table->setColumnWidth(i, width_column);
    }
    width += Table->columnWidth(i);
  }

  QObject::connect(Table, &QTableWidget::itemChanged, this, &Book::
   check_item);
}
```

Листинг 11. Код конструктора класса Book

2.2.2 Метод add_Page

Этот метод создан для добавления новой строки с данными человека в телефонную книгу. Метод принимает на вход указатель на экземпляр класса Page, в котором хранится вся информация о новом человеке и переменная типа данных Bool, с помощью которой регулируется будет ли добавляться этот указатель в вектор Pages. Также в конце выполнения метода связывается сигнал numbers_updated от указателя на новую страницу со слотом update класса Book. Код метода add_Page представлен на листинге 12.

```
void Book::add_Page(Page* new_page, bool Flag) {
  if (Flag) {
    Pages.push_back(new_page);
  }
  int new_Row = Table->rowCount();
  Table->insertRow(new_Row);
  QStringList Data_List = new_page->getAll_Data_Strings();
  for (int i = 0; i < Data_List.size(); i++) {
    Table->setItem(new_Row, i, new QTableWidgetItem(Data_List[i]));
    Table->resizeRowToContents(new_Row);
  }
  QGridLayout* layout_of_phones_item = new_page->getPhones_Layout();
  QWidget* phones_item = new QWidget();
  phones_item->setMinimumWidth(200);
  phones_item->setLayout(layout_of_phones_item);
  Table->setCellWidget(new_Row, Lheaders.size() - 1, phones_item);
  Table->resizeColumnToContents(Lheaders.size() - 1);
  Table->resizeRowToContents(new_Row);
  QObject::connect(new_page, &Page::numbers_updated, this, &Book::update
   );
}
```

Листинг 12. Код метода add_Page

2.2.3 Методы для получения значений переменных (геттеры)

В классе Book определены следующие методы для получения значений переменных: get_Table - метод для получения указателя на таблицу, содержащую инфор-

мацию телефонной книги, get_Width - метод для получения ширины всей таблицы, getPages - метод для получения вектора указателей на стриницы с информацией о людях. Коды этих методов представлены на листинге 13.

```
QTableWidget* Book::get_Table() { return Table; }
int Book::get_Width() { return width; }
QVector<Page*> Book::getPages() { return Pages; }
```

Листинг 13. Коды методов для получения значений переменных

2.2.4 слот add_Row

Данный слот осуществляет добавление новой страницы в конец вектора указателей на страницы с информацией о людях. Код этого слота представлен на листинге 14.

```
void Book::add_Row() { this->add_Page(new Page()); }
```

Листинг 14. Код слота add_Row

2.2.5 слот check_item

В этом слоте реализована обработка пользовательского ввода при помощи использования регулярных выражений. Метод check_item реализует обработку всех полей таблицы, кроме адреса и номеров телефонов (так как проверка правильности набора номера осуществляется в классе Page). Выбор способа обработки строки реализован с помощью определения номера столбца, в котором находится проверяемый элемент таблицы. Для полей фамилия, имя и отчество реализована одна проверка, так как они имеют одинаковые правилы написания. При обнаружении неправильного пользовательского ввода, появляется диалоговое окно QMessageBox для предупреждения пользователя о неправильном вводе. Код слота check_item представлен на листинге 15.

```
void Book::check_item(QTableWidgetItem* item) {
  switch (item->column())
  {
  case 0:
  case 1:
  case 2:
  {
    QString input = item->text().trimmed();
    QRegExp regex(QStringLiteral("^[A-AZЯЁ-][a-azяё-A-AZЯЁ-]*[
   -]?[0-9]*([ -]?[a-azяё-A-AZЯЁ-]*[ -]?[0-9]*)*"));
    bool isValid = regex.exactMatch(input);

    if (input.startsWith('-') || input.endsWith('-')) {
      isValid = false;
    }

    if (isValid) {
      item->setText(input);
      if (item->column() == 0) {
        Pages[item->row()]->setSurname(item->text());
      }
```

```
21        if (item->column() == 1) {
22          Pages[item->row()]->setName(item->text());
23        }
24        if (item->column() == 2) {
25          Pages[item->row()]->setPatronymic(item->text());
26        }
27      }
28      else {
29        QMessageBox::warning(Table->parentWidget(), Lerrors[static_cast<
    int>(Errors::incorrect_data)], Lerrors[static_cast<int>(Errors::
    incorrect_surname)]);
30        item->setText(Lvalues[static_cast<int>(Values::null)]);
31        if (item->column() == 0) {
32          Pages[item->row()]->setName(item->text());
33        }
34        if (item->column() == 1) {
35          Pages[item->row()]->setSurname(item->text());
36        }
37        if (item->column() == 2) {
38          Pages[item->row()]->setPatronymic(item->text());
39        }
40      }
41      break;
42    }
43    case 3:
44    {
45      Pages[item->row()]->setAddress(item->text());
46      break;
47    }
48    case 4:
49    {
50      QString input = item->text().trimmed();
51      if (item->text() == Lvalues[static_cast<int>(Values::null)]) {
52        return;
53      }
54      QRegExp regex(QStringLiteral("[0-9]+\\.[0-9]+\\.[0-9]+"));
55      bool isValid = regex.exactMatch(input);
56      if (isValid) {
57        QStringList digits = input.split(".");
58        QDate date = QDate(digits[2].toInt(), digits[1].toInt(), digits
    [0].toInt());
59        QDate today = QDate::currentDate();
60        if (date.isValid() and (date <= today)) {
61          item->setText(date.toString("dd.MM.yyyy"));
62          Pages[item->row()]->setDOB(QDate::fromString(item->text(), "dd.
    MM.yyyy"));
63          return;
64        }
65      }
66      else {
67        item->setText(Lvalues[static_cast<int>(Values::null)]);
68        Pages[item->row()]->setDOB(QDate());
69        QMessageBox::warning(Table->parentWidget(), Lerrors[static_cast<
    int>(Errors::incorrect_data)], Lerrors[static_cast<int>(Errors::
    incorrect_date)]);
70      }
```

```
71      break;
72    }
73    case 5:
74    {
75      QString input = item->text().trimmed();
76      if (item->text() == Lvalues[static_cast<int>(Values::null)]) {
77        return;
78      }
79      QRegExp regex(QStringLiteral("[a-zA-Z0-9\.]+[ ]?@[ ]?[a-zA-Z0-9\.]+"
     ));
80      bool isValid = regex.exactMatch(input);
81      if (isValid) {
82        item->setText(input);
83        Pages[item->row()]->setEmail(input);
84      }
85      else {
86        item->setText(Lvalues[static_cast<int>(Values::null)]);
87        Pages[item->row()]->setEmail(Lvalues[static_cast<int>(Values::null
     )]);
88        QMessageBox::warning(Table->parentWidget(), Lerrors[static_cast<
     int>(Errors::incorrect_data)], Lerrors[static_cast<int>(Errors::
     incorrect_email)]);
89      }
90      break;
91    }
92    default:
93      break;
94    }
95  }
```

Листинг 15. Код слота check_item

2.2.6 Слоты update и clear

Слот update создан для добавления возможности обновления всей телефонной книги. Этот слот прежде всего используется при добавлении нового номера телефона, так как если не обновить телефонную книгу, пользователь не увидит новый номер телефона.

С помощью слота clear реализовано удалений всей информации из телефонной книги и таблицы, которой отображается информация пользователю.

Коды этих слотов представлены на листинге 16.

```
1  void Book::update() {
2    Table->clearContents();
3    Table->setRowCount(0);
4    for (auto i : Pages) {
5      this->add_Page(i, false);
6    }
7  }
8
9  void Book::clear() {
10   Pages.clear();
11   Table->clearContents();
```

```
12   Table->setRowCount(0);
13 }
```

Листинг 16. Коды слотов update и clear

2.2.7 Слот delete_row

В данном слоте реализовано удаление выделенных таблице строк и их данных из телефонной книги. Сначала в слоте определяются номера выделенных строк, потом они сортируются в порядке убывания и каждая строка из этого списка удаляестя. Важно удалять строки именно в порядке убывания, так как иначе номера строк могут стать недействительными и при попытке удалить строку с этим индексом программа выдаст ошибку. Код этого слота представлен на листинге 17.

```
1  void Book::delete_row() {
2    QList<QTableWidgetSelectionRange> selectedRanges = Table->
       selectedRanges();
3    QSet<int> rowsToDelete;
4
5    for (const QTableWidgetSelectionRange& range : selectedRanges) {
6      for (int row = range.topRow(); row <= range.bottomRow(); ++row) {
7        rowsToDelete.insert(row);
8      }
9    }
10   QList<int> sortedRows = rowsToDelete.toList();
11   std::sort(sortedRows.begin(), sortedRows.end(), std::greater<int>());
12   for (int row : sortedRows) {
13     Table->removeRow(row);
14     Pages.remove(row);
15   }
16 }
```

Листинг 17. Код слота delete_row

2.2.8 Слот confirmExit

Этот слот обрабатывает нажатие пользователем на кнопку выхода. Пользователю показывается диалоговое окно, в котором предлагается перед выходом сохранить телефонную книгу в файл. Код слота представлен на листинге 19.

```
1  void Book::confirmExit(QApplication* app, QString Name) {
2    QMessageBox::StandardButton reply;
3    reply = QMessageBox::question(this->Table->parentWidget(), Lerrors[
       static_cast<int>(Errors::accept_exit)], Lerrors[static_cast<int>(
       Errors::save_exit)],
4      QMessageBox::Yes | QMessageBox::No | QMessageBox::Abort);
5    if (reply == QMessageBox::Yes) {
6      writeBookToJsonFile(this, Name);
7      app->quit();
8    }
9    if (reply == QMessageBox::No) {
10     app->quit();
11   }
```

```
12    if (reply == QMessageBox::Abort) {
13      return;
14    }
15  }
```

Листинг 18. Код слота confirmExit

2.3 Функции

2.3.1 Функция writeBookToJsonFile

Данная фукнция реализует сохранение телефонной книги в файл в формате Json. Для хранения информации из телефонной книги был выбран именно этот формат, так как он позволяет структурировать данные. Запись в файл осуществляется с помощью класса QFile из библиотеки Qt. Данная реализация позволяет пользователю выбирать название файла для сохранения, тем самым создавая несколько телефонных книг. Код этой функции представлен на листинге 19.

```
1  void writeBookToJsonFile(Book* book, const QString& fileName) {
2    QFile file;
3    if (not fileName.endsWith(".json")) {
4      file.setFileName(fileName + ".json");
5    }
6    else {
7      file.setFileName(fileName);
8    }
9    if (!file.open(QIODevice::WriteOnly | QIODevice::Text)) {
10     QMessageBox::warning(book->get_Table()->parentWidget(), Lerrors[
       static_cast<int>(Errors::error_save)], Lerrors[static_cast<int>(
       Errors::table_not_saved)]);
11     return;
12   }
13   QJsonArray jsonArray;
14   for (auto page : book->getPages()) {
15     QJsonObject jsonObject;
16     jsonObject[Lheaders[static_cast<int>(Headers::Surname)]] = page->
       getSurname();
17     jsonObject[Lheaders[static_cast<int>(Headers::Name)]] = page->
       getName();
18     jsonObject[Lheaders[static_cast<int>(Headers::Patronymic)]] = page->
       getPatronymic();
19     jsonObject[Lheaders[static_cast<int>(Headers::Address)]] = page->
       getAddress();
20     jsonObject[Lheaders[static_cast<int>(Headers::DOB)]] = page->getDOB
       ().toString("dd.MM.yyyy");
21     jsonObject[Lheaders[static_cast<int>(Headers::Email)]] = page->
       getEmail();
22     QJsonArray phoneNumbersArray = QJsonArray::fromStringList(
       QStringList(page->getPhone_Numbers().toList()));
23     QJsonArray phoneTypesArray = QJsonArray::fromStringList(page->
       getPhone_types_str().toList());
24     jsonObject["Phone_Numbers"] = phoneNumbersArray;
25     jsonObject["Phone_types"] = phoneTypesArray;
26     jsonArray.append(jsonObject);
```

```
27    }
28    QJsonDocument jsonDoc(jsonArray);
29    file.write(jsonDoc.toJson());
30    file.close();
31  }
```

Листинг 19. Код функции writeBookToJsonFile

Для функции writeBookToJsonFile, сохранение данных в JSON-формате происходит следующим образом:

1. Корневым элементом является массив jsonArray, который содержит объекты для каждой страницы книги.
2. Каждый объект в массиве представляет одну страницу (или запись) и содержит следующие ключи и значения:
   - Surname: Фамилия (строка)
   - Name: Имя (строка)
   - Patronymic: Отчество (строка)
   - Address: Адрес (строка)
   - DOB: Дата рождения в формате "dd.MM.yyyy"(строка)
   - Email: Электронная почта (строка)
   - Phone_Numbers: Массив номеров телефонов (массив строк)
   - Phone_types: Массив типов телефонов (массив строк)

Номера телефонов и их типы сохраняются как массивы, что позволяет зранить несколько значений для одного человека. Данная структура сохранения данных позволяет в будущем добавлять новые поля без изменения базового формата. Не были сериализованы данные о возможности редактирования таблицы, потому что запрет этой возможности контролируется только с помощью таблицы для представления данных пользователю.

2.3.2 Функция readBookFromJsonFile

Эта функция реализует считывание данных из файла формата Json в телефонную книгу. Пользователь также может выбрать файл из которого будет считываться информация о телефонной книге. Код этой функции представлен на листинге 20.

```
1  bool readBookFromJsonFile(Book* book, const QString& fileName) {
2    QFile file(fileName);
3    if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
4      QMessageBox::warning(book->get_Table()->parentWidget(), Lerrors[
       static_cast<int>(Errors::error_open)], Lerrors[static_cast<int>(
       Errors::not_open)]);
5      return false;
6    }
7    QByteArray fileData = file.readAll();
8    file.close();
```

```
9   QJsonDocument jsonDoc = QJsonDocument::fromJson(fileData);
10  if (!jsonDoc.isArray()) {
11    QMessageBox::warning(book->get_Table()->parentWidget(), Lerrors[
     static_cast<int>(Errors::error_open)], Lerrors[static_cast<int>(
     Errors::bad_json)]);
12    return false;
13  }
14  QJsonArray jsonArray = jsonDoc.array();
15  for (const QJsonValue& value : jsonArray) {
16    if (!value.isObject()) {
17      continue;
18    }
19    QJsonObject obj = value.toObject();
20    Page* page = new Page();
21    page->setSurname(obj[Lheaders[static_cast<int>(Headers::Surname)]].
     toString());
22    page->setName(obj[Lheaders[static_cast<int>(Headers::Name)]].
     toString());
23    page->setPatronymic(obj[Lheaders[static_cast<int>(Headers::
     Patronymic)]].toString());
24    page->setAddress(obj[Lheaders[static_cast<int>(Headers::Address)]].
     toString());
25    page->setDOB(QDate::fromString(obj[Lheaders[static_cast<int>(Headers
     ::DOB)]].toString(), "dd.MM.yyyy"));
26    page->setEmail(obj[Lheaders[static_cast<int>(Headers::Email)]].
     toString());
27    QJsonArray phoneNumbersArray = obj["Phone_Numbers"].toArray();
28    QVector<QString> new_phone_numbers;
29    for (const QJsonValue& phoneNumber : phoneNumbersArray) {
30      new_phone_numbers.append(phoneNumber.toString());
31    }
32    page->setPhone_Numbers(new_phone_numbers);
33    QJsonArray phoneTypesArray = obj["Phone_types"].toArray();
34    QVector<int> new_phone_types;
35    for (const QJsonValue& phoneType : phoneTypesArray) {
36      new_phone_types.append(phoneType.toInt());
37    }
38    page->setPhone_types(new_phone_types);
39    book->add_Page(page);
40  }
41  book->update();
42  return true;
43 }
```

Листинг 20. код функции readBookFromJsonFile

2.3.3 Функции для выключения редактирования телефонной книги

Для выключения редактирования телефонной книги создано три фукнции: setWidgetsEnabled, set_numbers_const и disableWidgetsInLastColumn.

Функции setWidgetsEnabled и set_numbers_const созданы для запрета изменения виджетов в последнем столбце таблицы (номера телефонов). Их коды представ-

лены на листингах 21 и 22.

Функция disableWidgetsInLastColumn отключает позможность взаимодействия пользователя с каждым виджетом находящимся в столбце с номерами телефонов, то есть делает невозможным изменение номеров телефонов, их типов и добавления новых номеров. Код этой функции представлен на листинге 23.

```
void setWidgetsEnabled(QGridLayout* layout) {
  for (int i = 0; i < layout->rowCount(); ++i) {
    for (int j = 0; j < layout->columnCount(); ++j) {
      QWidget* widget = layout->itemAt(i * layout->columnCount() + j)->
   widget();
      if (widget) {
        widget->setEnabled(widget->isEnabled() ? false : true);
      }
    }
  }
}
```

Листинг 21. Код функции setWidgetsEnabled

```
void set_numbers_const(QTableWidget* Table) {
  for (int i = 0; i < Table->rowCount(); i++) {
    setWidgetsEnabled(qobject_cast<QGridLayout*>(Table->itemAt(i,
   Headers.size() - 1)->tableWidget()->layout()));
  }
}
```

Листинг 22. Код функции set_numbers_const

```
void disableWidgetsInLastColumn(QTableWidget* table) {
  QWidget* layout_widget1;
  QWidget* layout_widget2;
  QWidget* layout_widget3;
  QGridLayout* grid_layout;
  for (int i = 0; i < table->rowCount(); i++) {
    grid_layout = qobject_cast<QGridLayout*>(table->cellWidget(i, table
   ->columnCount() - 1)->layout());
    for (int j = 0; j < grid_layout->rowCount() - 1; j++) {
      layout_widget1 = grid_layout->itemAtPosition(j, 0)->widget();
      layout_widget2 = grid_layout->itemAtPosition(j, 1)->widget();

      layout_widget1->setEnabled(not layout_widget1->isEnabled());
      layout_widget2->setEnabled(not layout_widget2->isEnabled());
    }
    layout_widget3 = grid_layout->itemAtPosition(grid_layout->rowCount()
    - 1, 0)->widget();
    layout_widget3->setEnabled(not layout_widget3->isEnabled());

```

Листинг 23. Код функции disableWidgetsInLastColumn

2.3.4 Функция search

функция search реализует поиск определённого значения в таблице. Пользователь задаёт строку которую хочет найти и поле по которому будет произведён поиск.

Данная фукнция возвращает указатель на новую таблицу QTableWidget, в которой уже будут находится только те строчки, в которых найдена строка заданная пользователем, при этом старая таблица не удаляется, пользователь может к ней вернутся нажав кнопку "Отмена". Код этой функции представлен на листинге 24.

```
QTableWidget* search(Book* book, int index_of_combo, const QString&
    request) {
  if (book->get_Table() == nullptr || index_of_combo < 0 ||
   index_of_combo >= book->get_Table()->columnCount()) {
    return nullptr;
  }
  Book* result_book = new Book(book->get_Table()->parentWidget());

  book->get_Table()->setGeometry(QRect(0, 31, 1200, 770));
  book->get_Table()->setHorizontalHeaderLabels(Lheaders);
  for (Page* p : book->getPages()) {
    bool flag = false;
    switch (index_of_combo)
    {
    case 0:
      flag = p->getSurname().contains(request);
      break;
    case 1:
      flag = p->getName().contains(request);
      break;
    case 2:
      flag = p->getPatronymic().contains(request);
      break;
    case 3:
      flag = p->getAddress().contains(request);
      break;
    case 4:
      flag = p->getDOB().toString("dd.MM.yyyy").contains(request);
      break;
    case 5:
      flag = p->getEmail().contains(request);
      break;
    case 6:
      for (QString number : p->getPhone_Numbers()) {
        flag = number.contains(request);
        if (flag) {
          break;
        }
      }
      break;
    default:
      break;
    }
    if (flag) {
      result_book->add_Page(p);
    }
  }
  return result_book->get_Table();
}
```

Листинг 24. Код функции search

2.4 Хранение константных значений

Для хранения константных значений, которые многократно используются в программе, были созданы перечисления и соответствующие им списки строк.

Для хранения названий столбцов таблицы используется перечисление Headers и список строк Lheaders. Их коды представлены на листингах 25 и 26 соответственно.

```
enum Headers
{ Surname,
  Name,
  Patronymic,
  Address,
  DOB,
  Email,
  Phone_Numbers };
```

Листинг 25. Код перечисления Headers

```
QStringList Lheaders = {
  QStringLiteral("Фамилия"),
  QStringLiteral("Имя"),
  QStringLiteral("Отчество"),
  QStringLiteral("Адрес"),
  QStringLiteral("Дата рождения"),
  QStringLiteral("E-mail"),
  QStringLiteral("Номера телефонов")
};
```

Листинг 26. Код списка строк LHeaders

Для хранения строк, использующихся в ячейках таблицы, было создано перечисление Values и список строк LValues. Их коды представлены на листингах 27 и 28.

```
enum Values
{ null,
  home,
  work,
  personal };
```

Листинг 27. Код перечисления Headers

```
QStringList Lvalues = {
  QStringLiteral("Не задано"),
  QStringLiteral("Домашний"),
  QStringLiteral("Рабочий"),
  QStringLiteral("Личный")
};
```

Листинг 28. Код списка строк LHeaders

Для хранения строк, которые используются для общения с пользователем, используется перечисление Errors и список строк Lerrors. Их коды представлены на листингах 29 и 30 соответственно.

```
enum Errors
{ incorrect_data,
```

```
  incorrect_phone,
  incorrect_surname,
  incorrect_date,
  incorrect_email,
  error_save,
  table_not_saved,
  not_open,
  not_opened,
  bad_json,
  save_file,
  open_file,
  accept_exit,
  save_exit,
  error_open };
```

Листинг 29. Код перечисления Headers

```
QStringList Lerrors = {
  QStringLiteral("Некорректные данные"),
  QStringLiteral("Не верный номер телефона. Номер телефона должен быть написан
   по одному из следующих образцов: +7 812 1234567, 8(800)123-1212, + 38
   (032) 123 11 11. Строка автоматически заменена на Не\" задано\""),
  QStringLiteral("Фамилия, имя и отчество должны начинаться с заглавной буквы,
   могут содержать только буквы, цифры, дефис и пробел, а также не должны
   оканчиваться на дефис. Строка автоматически заменена на Не\" задано\""),
  QStringLiteral("Не верная дата. Строка автоматически заменена на Не\" задано\"
   "),
  QStringLiteral("Не верный E-mail. Строка автоматически заменена на Не\"
   задано\""),
  QStringLiteral("Ошибка сохранения"),
  QStringLiteral("Таблица не была сохранена"),
  QStringLiteral("Не удалось открыть файл"),
  QStringLiteral("Неверный формат JSON"),
  QStringLiteral("Сохранить файл"),
  QStringLiteral("Открыть файл"),
  QStringLiteral("Подтвердите выход"),
  QStringLiteral("Сохранить файл перед выходом?"),
  QStringLiteral("Ошибка открытия файла")
};
```

Листинг 30. Код списка строк LHeaders

3 Тестирование приложения

После запуска приложения пользователь видит начальное окно программы (Рис. 1).

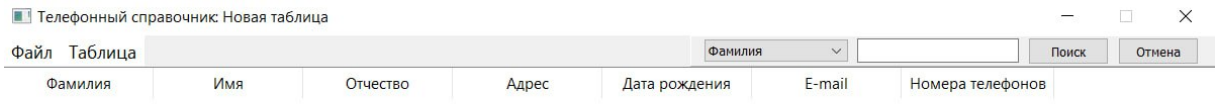

Рис. 1. Программа для отображения графика кубического сплайна

В верхнем меню приложения, есть две вкладки «Файл» и «Таблица» (Рис. 2 и Рис. 3 соответственно). Во вкладке «Файл» находятся действия связанные с сохранением и загрузкой телефонного справочника, а также кнопка выхода из приложения. Во вкладке «Таблица» находятся действия для редактирования таблицы и кнопки для переключения режимов работы с таблицей («Переключить сортировку» и «Разрешить редактирование»).

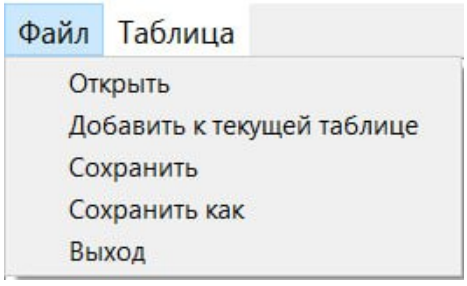

Рис. 2. Вкладка «Файл»

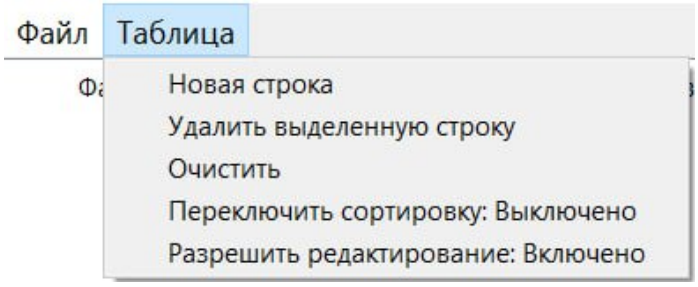

Рис. 3. Вкладка «Таблица»

При нажатии пользователем кнопки «Новая строка» во вкладке «Таблица» в телефонном справочнике появляется новая пустая запись (Рис. 4)

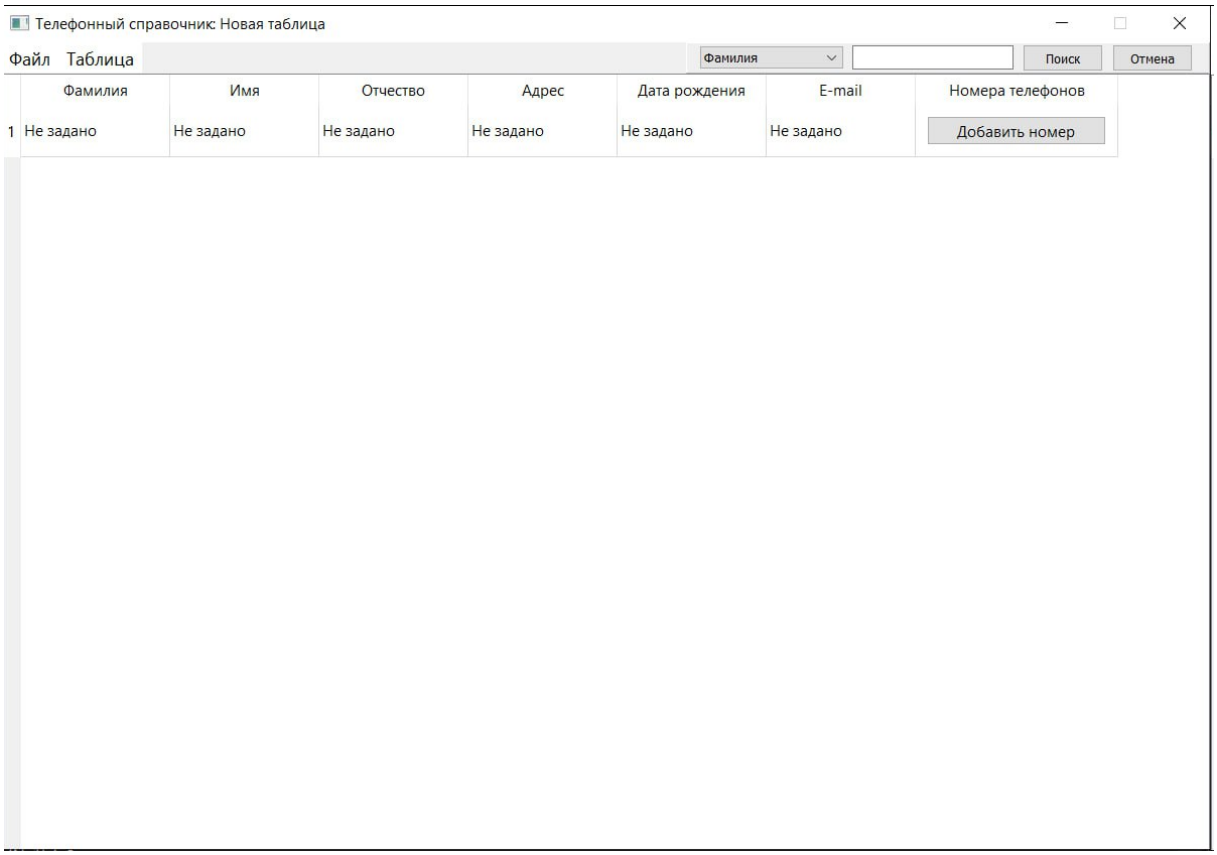

Рис. 4. Добавление строки

Также пользователь может открыть файл с уже имеющимся телефонным справочником с помощью нажатия на кнопку «Открыть» во вкладке «Файл». После нажатия на кнопку пользователю будет предоставлено окно, в котром он сможет выбрать интересующий его файл (Рис. 5). Если файл повреждён или имеет неправильную запись информации внутри, то пользователю показывается предупреждение о том, что файл не удалось открыть (Рис. 6).

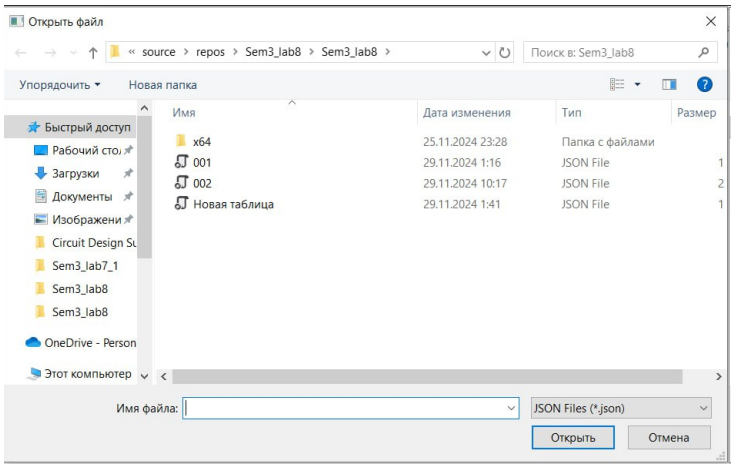

Рис. 5. Выбор файла

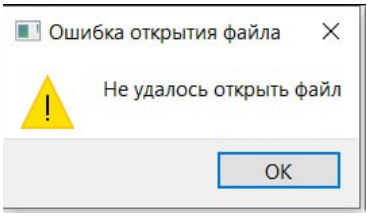

Рис. 6. Предупреждение

После удачного открытия файла в таблице начинает отображаться информация из этого файла (Рис. 7).

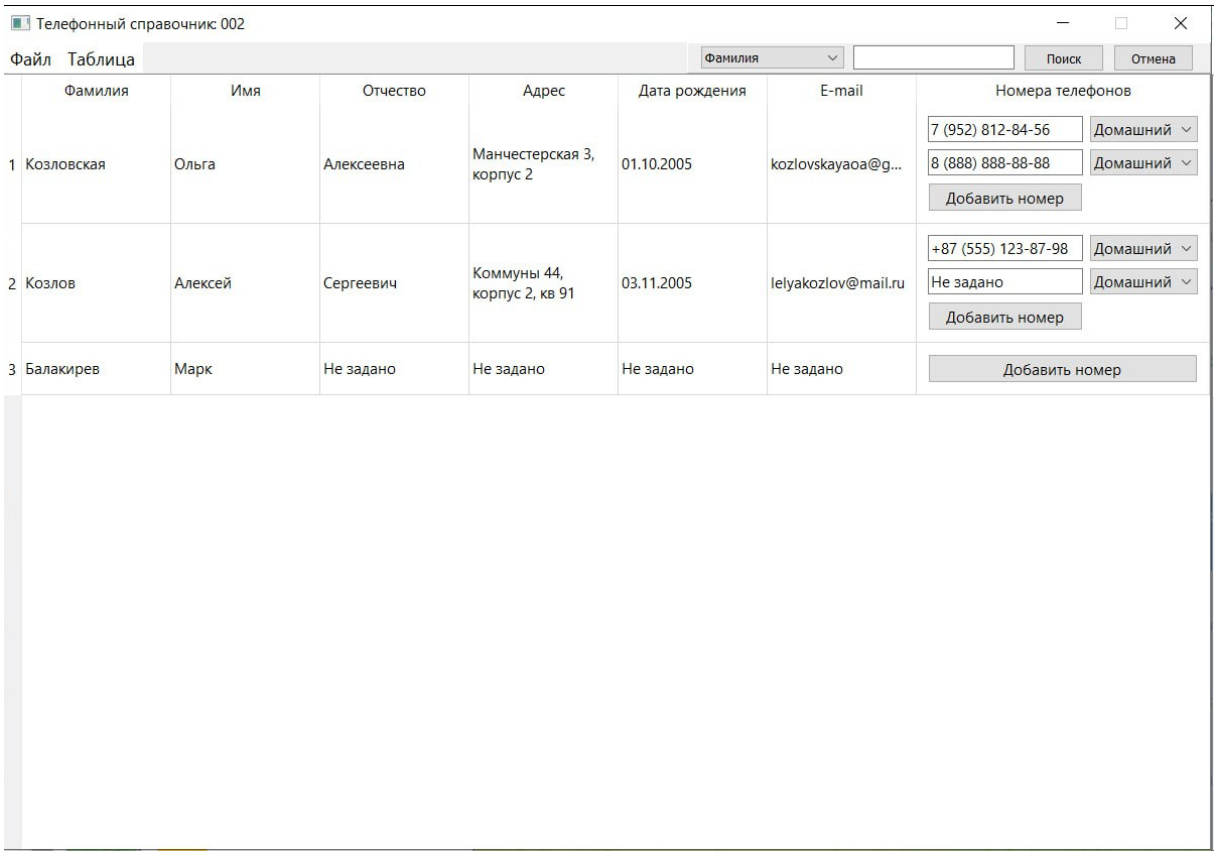

Рис. 7. Таблица после открытия файла

Для добавления номера телефона пользователь должен нажать на кнопку «Добавить номер» в последнем столбце и после этого в этой ячейке появится строка для набора номера и список, в котором можно выбрать тип номера телефона (Рис. 8).

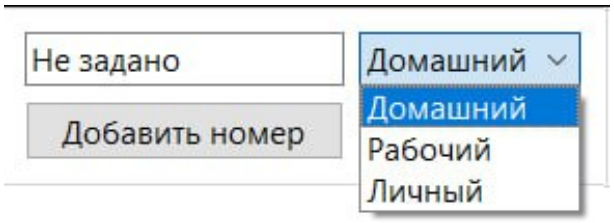

Рис. 8. Виджет для записи номера телефона

В приложении также реализован поиск определённой строки в выбранном пользователем столбце. Виджеты его использование расположены в правом верхнем углу меню (Рис. 9). В выпадающем списке пользователь выбирает поле в котором он хочет найти строку, далее он пишет в строку для ввода то, что он хчоет найти и нажимает кнопку «Поиск». В результате в таблице оказываются только те строчки, в которых была найдена строка пользователя (Рис. 9).

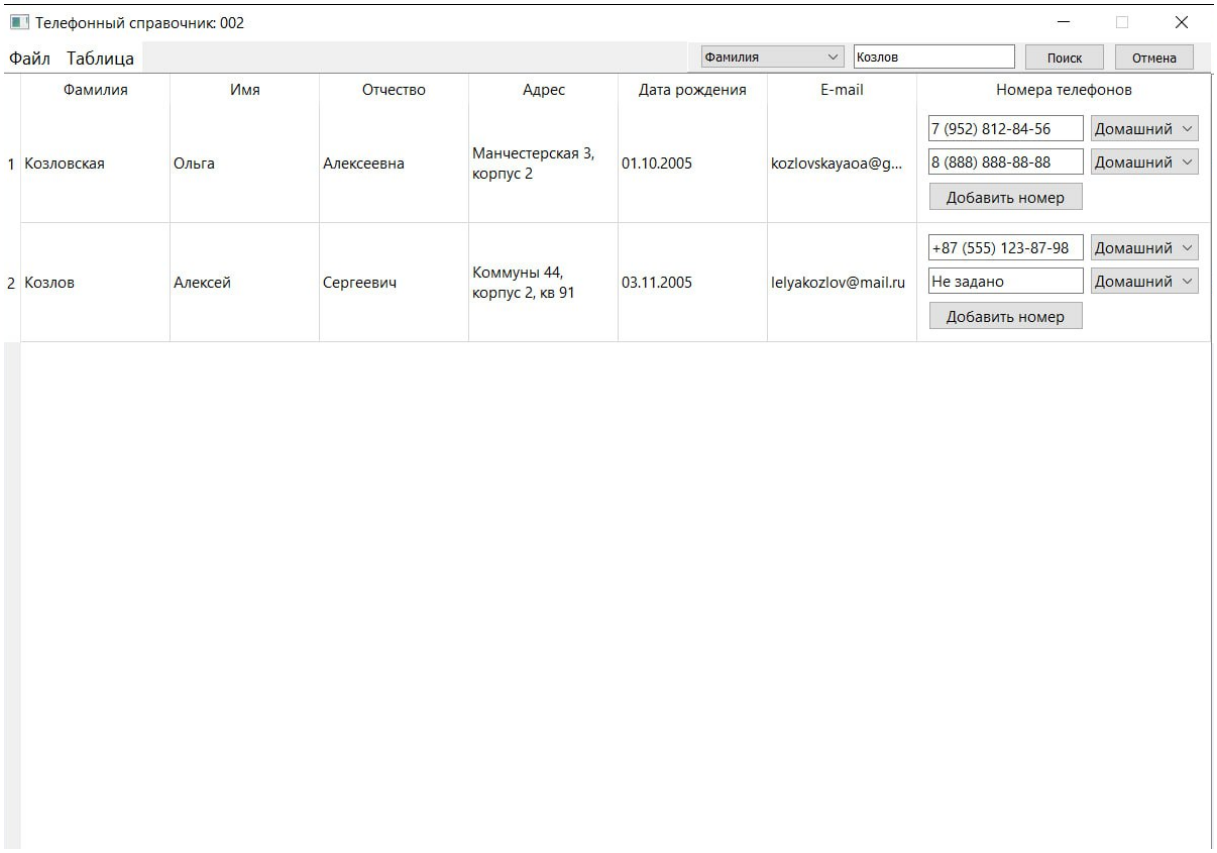

Рис. 9. Таблица после поиска

При нажатии кнопки «Отмена» таблица вернётся в исходное состояние до поиска. Пользователь может удалять выделенные строки в таблице из телефонного справочника, причём выделенные строки могут находится в таблице произвольным образом. Выделение строк представлено на Рис. 10, а результат удаления выделенных строк на Рис. 11.

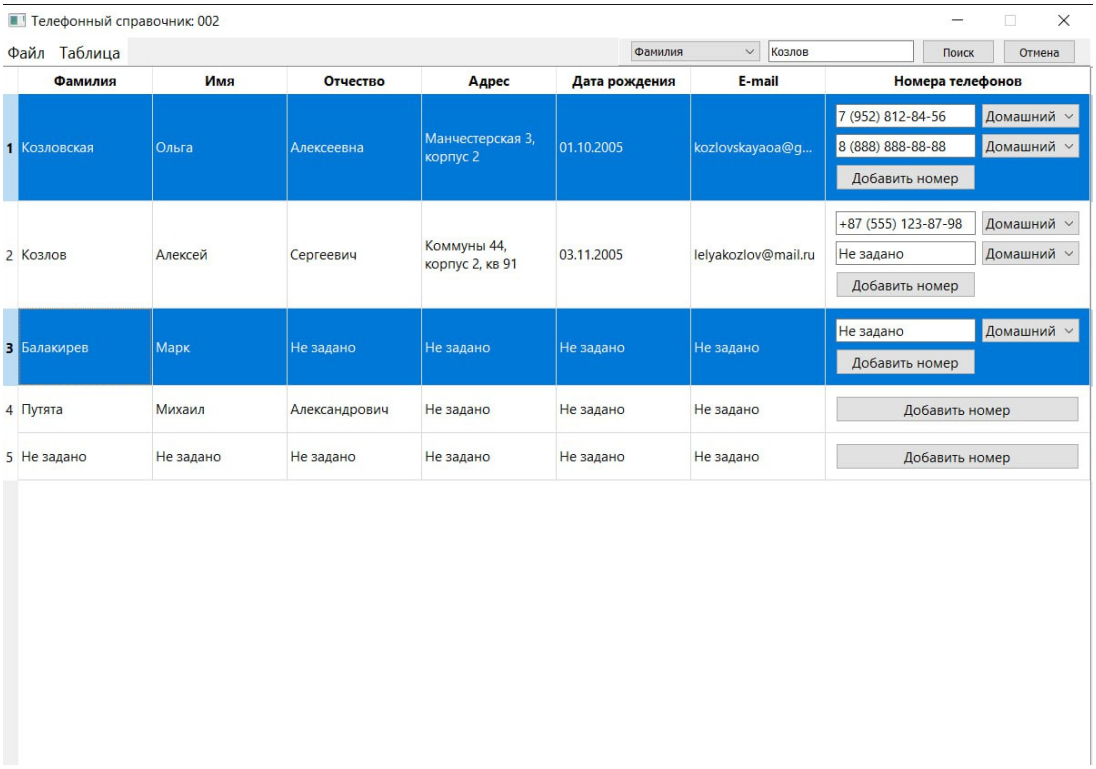

Рис. 10. Выделение строк в таблице

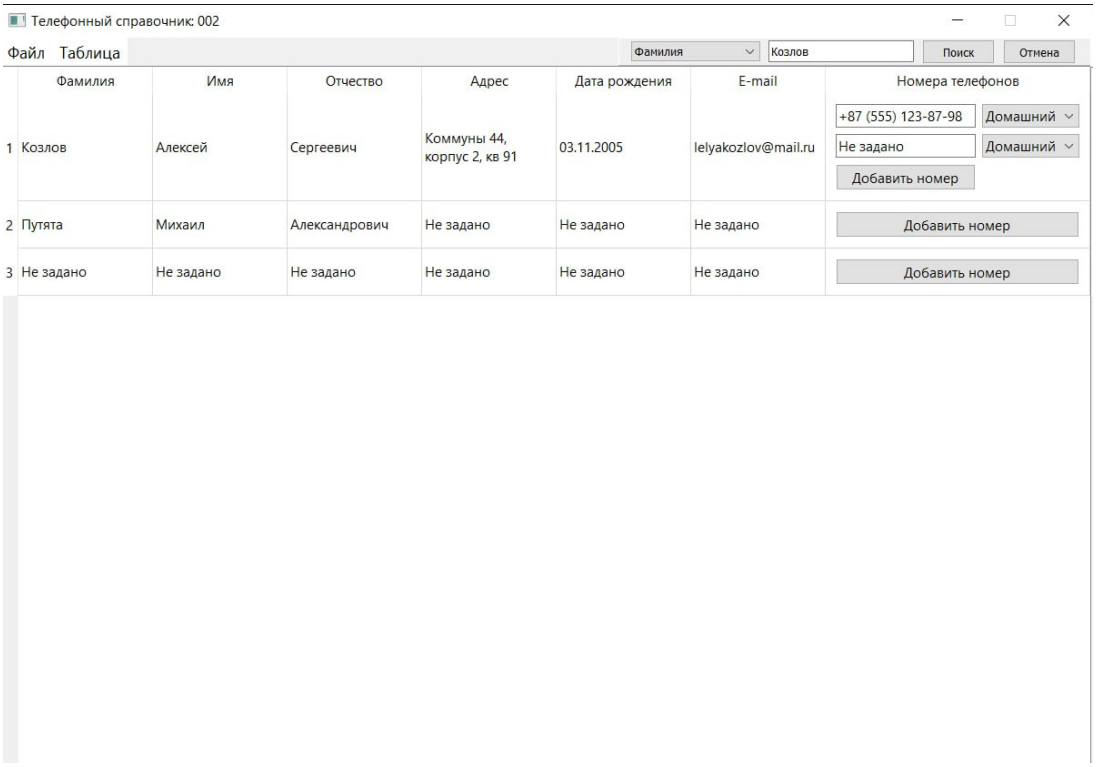

Рис. 11. Удаление выделенных строк

В приложении реализована сортировка данных в таблице. Для включения этой

функции во вкладке «Таблица» верхнего меню нужно включить кнопку «Переключить сортировку». Тогда, нажав на заголовок любого столбца, можно будет включать сортировку по значениям этого столбца. На Рис. 12 представлена таблица со включённой сортировкой по столбцу «Имя».

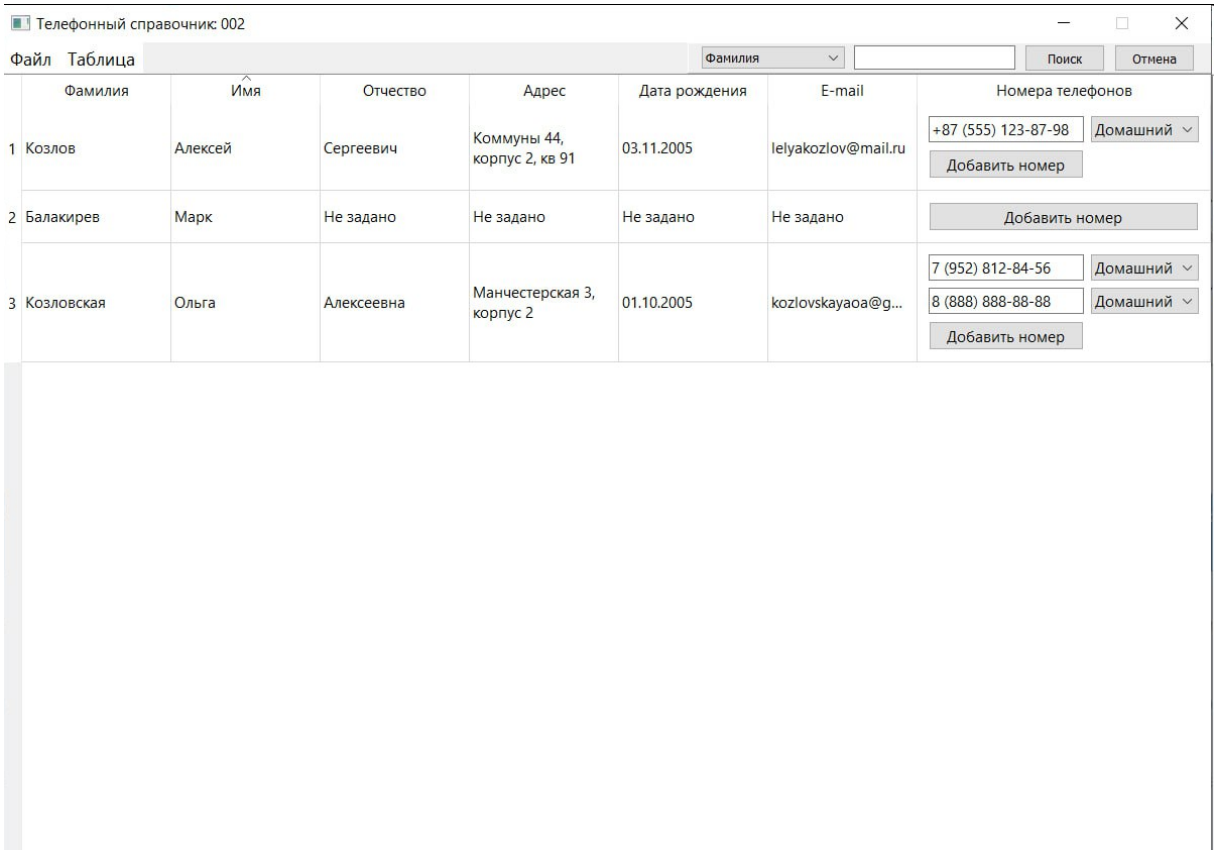

Рис. 12. Сортировка по полю «Имя»

После перевода кнопки «Включить редактирование» во вкладке «Таблица» в выключенное положение пользоатель не сможет редактировать созданную телефонную книгу - все поля таблицы станут неизменяемыми, а последний столбик, содержащий в себе номера телефонов, станет окрашен в серый цвет и также будет неизменяем (Рис. 13).

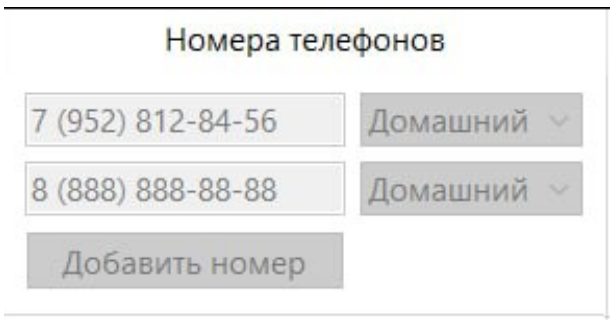

Рис. 13. Поле с номерами при выключении редактирования

При нажатии пользователем на клавишу «Выход» во вкладке «Файл» будет показано диалоговое окно с предложением сохранить файл (Рис. 14). При нажатии кнопки «Yes» телефонная книга будет сохранена в тот же файл, из которого она была открыта, если же телефонная книга создавалась как новая таблица, она будет сохранена в файл с названием «Новая таблица», и после этого программа закроется. При нажатии на кнопку «No» программа закроется, а при нажатии на кнопку «Abort» пользователь вернётся в режим редактирования телефонной книги.

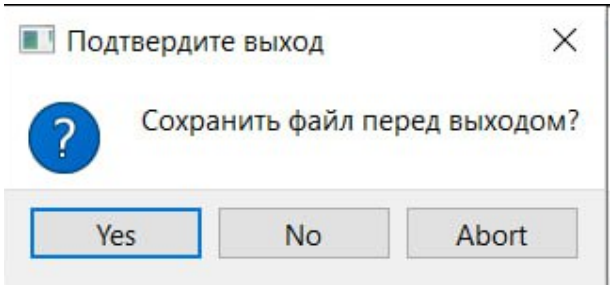

Рис. 14. Предложение сохранить файл

Заключение

В результате работы все поставленные задачи были выполнены, а также был добавлен дополнительный функционал в приложение (добавление данных из файла в текущую телефонную книгу, переключение редактирования данных). Реализованное приложение работает корректно и привильно выполняет все функции.

Код всего приложения состоит из 800 строчек.

Работа была выполнена в среде разработки Visual Studio 2022. Использовался стандарт языка ISO C++ 14 и фреймворк Qt версии 5.12.2.

Для улучшение приложения можно сделать возможным открытие сразу нескольких телефонных книг и дать пользователю возможность создавать свои поля в таблице.

Опыт полученный во время написание этой курсовой работы будет полезен в дальнейшей профессиональной деятельности, так как зачастую очень важно правильно и удобно предоставлять пользователю данные.

Приложение А. Исходный код класса Page

А.1 Исходный код файла Page.h

```
#pragma once

#include <QString>
#include <QDate>
#include <QVector>
#include <QStandardItemModel>
#include <QGridLayout>
#include <QlineEdit>
#include <QComboBox>
#include <QPushButton>

class Book;

class Page : public QWidget {
  Q_OBJECT;
private:
  QString Surname;
  QString Name;
  QString Patronymic;
  QString Address;
  QDate DOB;
  QString Email;
  QVector<QString> Phone_Numbers;
  QVector<int> Phone_types;
public:
  Page();
  Page(QString surname_, QString name_, QString patronymic_, QString
    address_, QDate dob_, QString email_, QVector<QString> phone_numbers_
    , QVector<int> phone_types_);
  QString getSurname() const;
  QString getName() const;
  QString getPatronymic() const;
  QString getAddress() const;
  QDate getDOB() const;
  QString getEmail() const;
  QVector<QString> getPhone_Numbers() const;
  QVector<int> getPhone_types();
  QVector<QString> getPhone_types_str();
  QStringList getAll_Data_Strings() const;
  QGridLayout* getPhones_Layout();
  void setSurname(QString surname_);
  void setName(QString name_);
  void setPatronymic(QString patronymic_);
  void setAddress(QString address_);
  void setDOB(QDate dob_);
  void setEmail(QString email_);
  void setPhone_Numbers(QVector<QString> phone_numbers_);
  void setPhone_types(QVector<int> new_phone_types);
  void check_line(QLineEdit* line, int n);
  void check_combo(int n, int index);
signals:
```

```
50   void numbers_updated();
51 public slots:
52   void add_row_to_numbers();
53 };
```

A.2 Исходный код файла Page.cpp

```
#include "Page.h"
#include "Book.h"
#include "Header.h"

Page::Page() {

}

Page::Page(QString surname_, QString name_, QString patronymic_, QString
    address_, QDate dob_, QString email_, QVector<QString>
   phone_numbers_, QVector<int> phone_types_) {
  Surname = surname_;
  Name = name_;
  Patronymic = patronymic_;
  Address = address_;
  DOB = dob_;
  Email = email_;
  Phone_Numbers = phone_numbers_;
  Phone_types = phone_types_;
}

QString Page::getSurname() const { return Surname; }
QString Page::getName() const { return Name; }
QString Page::getPatronymic() const { return Patronymic; }
QString Page::getAddress() const { return Address; }
QDate Page::getDOB() const { return DOB; }
QString Page::getEmail() const { return Email; }
QVector<QString> Page::getPhone_Numbers() const { return Phone_Numbers;
   }
QVector<int> Page::getPhone_types() { return Phone_types; }

QStringList Page::getAll_Data_Strings() const{
  QStringList Out;
  Out << (Surname.isEmpty() ? Lvalues[static_cast<int>(Values::null)] :
   Surname);
  Out << (Name.isEmpty() ? Lvalues[static_cast<int>(Values::null)] :
   Name);
  Out << (Patronymic.isEmpty() ? Lvalues[static_cast<int>(Values::null)]
    : Patronymic);
  Out << (Address.isEmpty() ? Lvalues[static_cast<int>(Values::null)] :
   Address);
  Out << (DOB.isNull() ? Lvalues[static_cast<int>(Values::null)] : DOB.
   toString("dd.MM.yyyy"));
  Out << (Email.isEmpty() ? Lvalues[static_cast<int>(Values::null)] :
   Email);
  return Out;
}

QGridLayout* Page::getPhones_Layout() {
  QGridLayout* layout_phones = new QGridLayout();
  QStringList str_types = { Lvalues[static_cast<int>(Values::home)],
   Lvalues[static_cast<int>(Values::work)], Lvalues[static_cast<int>(
   Values::personal)] };
  for (int i = 0; i < Phone_Numbers.size(); i++) {
```

```
44
45      QLineEdit* line = new QLineEdit();
46      QComboBox* types = new QComboBox();
47
48      line->setText(Phone_Numbers[i]);
49      types->addItems(str_types);
50      types->setCurrentIndex(Phone_types[i]);
51
52      layout_phones->addWidget(line, i, 0);
53      QObject::connect(line, &QLineEdit::editingFinished, [line, i, this
        ]() {
54          check_line(line, i);
55          });
56      layout_phones->addWidget(types, i, 1);
57      QObject::connect(types, QOverload<int>::of(&QComboBox::
        currentIndexChanged), [i, this](int index) {
58          check_combo(i, index);
59          });
60    }
61    QPushButton* add_number_button = new QPushButton(QStringLiteral("
       Добавить номер"));
62    layout_phones->addWidget(add_number_button);
63    QObject::connect(add_number_button, &QPushButton::clicked, this, &Page
       ::add_row_to_numbers);
64    return layout_phones;
65  }
66
67
68  void Page::setSurname(QString surname_) { Surname = surname_; }
69  void Page::setName(QString name_) { Name = name_; }
70  void Page::setPatronymic(QString patronymic_) { Patronymic = patronymic_
       ; }
71  void Page::setAddress(QString address_) { Address = address_; }
72  void Page::setDOB(QDate dob_) { DOB = dob_; }
73  void Page::setEmail(QString email_) { Email = email_; }
74  void Page::setPhone_Numbers(QVector<QString> phone_numbers_) {
       Phone_Numbers = phone_numbers_; }
75
76
77  void Page::add_row_to_numbers() {
78    Phone_Numbers.push_back(Lvalues[static_cast<int>(Values::null)]);
79    Phone_types.push_back(0);
80    emit numbers_updated();
81  }
82
83  void Page::check_line(QLineEdit* line, int n) {
84    QString input = line->text().trimmed();
85    if (line->text() == QString(Lvalues[static_cast<int>(Values::null)]))
       {
86      return;
87    }
88    QRegExp regex("\\+?[ ]?[0-9][0-9]?[0-9]?[ ]?(\\(([0-9]{3}\\))
       |([0-9]{3}))[ ]?[0-9]{3}(([0-9]{4})|([-][0-9]{2}[ -]?[0-9]{2})|(
       [0-9]{2} [0-9]{2}))");
89    bool isValid = regex.exactMatch(input);
90    if (not isValid and line->text() != QString(Lvalues[static_cast<int>(
```

```
    Values::null)])) {
91    line->setText(Lvalues[static_cast<int>(Values::null)]);
92    Phone_Numbers[n] = Lvalues[static_cast<int>(Values::null)];
93    QMessageBox::warning(this, Lerrors[static_cast<int>(Errors::
    incorrect_data)], Lerrors[static_cast<int>(Errors::incorrect_phone)])
    ;
94
95
96  }
97  else {
98    QString number_removed = line->text().remove("+").remove(" ").remove
    ("-").remove("(").remove(")");
99
100   QString lasts = number_removed.right(10);
101   QString firsts = number_removed.left(number_removed.size() - 10);
102   QString out = firsts + " (" + lasts.left(3) + ") " + lasts.mid(3, 3)
    + "-" + lasts.mid(6, 2) + "-" + lasts.mid(8, 2);
103   if (line->text().startsWith("+")) {
104     out = "+" + out;
105   }
106   line->setText(out);
107   Phone_Numbers[n] = out;
108 }
109 }
110
111 void Page::check_combo(int n, int index) {
112   Phone_types[n] = index;
113 }
114
115 QVector<QString> Page::getPhone_types_str() {
116   QVector<QString> out;
117   for (auto i : Phone_types) {
118     out.push_back(QString::number(i));
119   }
120   return out;
121 }
122
123 void Page::setPhone_types(QVector<int> new_phone_types) {
124   Phone_types = new_phone_types;
125 }
```

Приложение Б. Исходный код класса Book

Б.2 Исходный код файла Book.h

```
#pragma once
#include "Page.h"
#include <QTableWidget>
#include <QMessageBox>
#include <QRegExp>
#include <QLineEdit>

#include <QJsonDocument>
#include <QJsonObject>
#include <QJsonArray>
#include <QFile>
#include <QFileDialog>
#include <QApplication>

class Book : public QObject{
  Q_OBJECT;
private:
  QVector<Page*> Pages;
  QTableWidget* Table;
  int width;
public:
  Book(QWidget* parent_);
  void add_Page(Page* new_page, bool Flag = true);

  QTableWidget* get_Table();
  int get_Width();
  QVector<Page*> getPages();
  void clear();

public slots:
  void add_Row();
  void check_item(QTableWidgetItem* item);
  void update();
  void delete_row();
  void confirmExit(QApplication* app, QString Name);
};

void writeBookToJsonFile(Book* book, const QString& fileName);
bool readBookFromJsonFile(Book* book, const QString& fileName);

QString getSaveFilePath(QWidget* parent = nullptr);
QString getOpenFilePath(QWidget* parent = nullptr);

void setWidgetsEnabled(QGridLayout* layout);
void set_numbers_const(QTableWidget* Table);
void disableWidgetsInLastColumn(QTableWidget* tableWidget);
QTableWidget* search(Book* book, int index_of_combo, const QString&
   request);
```

Б.2 Исходный код файла Book.cpp

```
#include "Book.h"
#include "Header.h"

int width_column = 148;



Book::Book(QWidget* parent_) {
  Table = new QTableWidget(0, Lheaders.size(), parent_);
  Table->setHorizontalHeaderLabels(Lheaders);
  Table->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
  Table->resizeColumnsToContents();
  width = 0;
  for (int i = 0; i < Lheaders.size(); i++) {
    if (Table->columnWidth(i) < width_column) {
      Table->setColumnWidth(i, width_column);
    }
    width += Table->columnWidth(i);
  }

  QObject::connect(Table, &QTableWidget::itemChanged, this, &Book::
   check_item);
}

void Book::add_Page(Page* new_page, bool Flag) {
  if (Flag) {
    Pages.push_back(new_page);
  }


  int new_Row = Table->rowCount();
  Table->insertRow(new_Row);

  QStringList Data_List = new_page->getAll_Data_Strings();
  for (int i = 0; i < Data_List.size(); i++) {
    Table->setItem(new_Row, i, new QTableWidgetItem(Data_List[i]));
    Table->resizeRowToContents(new_Row);
  }

  QGridLayout* layout_of_phones_item = new_page->getPhones_Layout();
  QWidget* phones_item = new QWidget();
  phones_item->setMinimumWidth(200);
  phones_item->setLayout(layout_of_phones_item);
  Table->setCellWidget(new_Row, Lheaders.size() - 1, phones_item);
  Table->resizeColumnToContents(Lheaders.size() - 1);
  Table->resizeRowToContents(new_Row);
  QObject::connect(new_page, &Page::numbers_updated, this, &Book::update
   );
}

QTableWidget* Book::get_Table() {
  return Table;
}

```

```
int Book::get_Width() {
  return width;
}


void Book::add_Row() {
  this->add_Page(new Page());
}

void Book::check_item(QTableWidgetItem* item) {
  switch (item->column())
  {
  case 0:
  case 1:
  case 2:
  {
    QString input = item->text().trimmed();
    QRegExp regex(QStringLiteral("^[A-AZЯЁ-][a-azяё-A-AZЯЁ-]*[ -]?[0-9]*([ -]?[a-azяё-A-AZЯЁ-]*[ -]?[0-9]*)*"));
    bool isValid = regex.exactMatch(input);

    if (input.startsWith('-') || input.endsWith('-')) {
      isValid = false;
    }

    if (isValid) {
      item->setText(input);
      if (item->column() == 0) {
        Pages[item->row()]->setSurname(item->text());
      }
      if (item->column() == 1) {
        Pages[item->row()]->setName(item->text());
      }
      if (item->column() == 2) {
        Pages[item->row()]->setPatronymic(item->text());
      }
    }
    else {
      QMessageBox::warning(Table->parentWidget(), Lerrors[static_cast<int>(Errors::incorrect_data)], Lerrors[static_cast<int>(Errors::incorrect_surname)]);
      item->setText(Lvalues[static_cast<int>(Values::null)]);
      if (item->column() == 0) {
        Pages[item->row()]->setName(item->text());
      }
      if (item->column() == 1) {
        Pages[item->row()]->setSurname(item->text());
      }
      if (item->column() == 2) {
        Pages[item->row()]->setPatronymic(item->text());
      }

    }
    break;
  }
  case 3:
```

```
106   {
107     Pages[item->row()]->setAddress(item->text());
108     break;
109   }
110   case 4:
111   {
112     QString input = item->text().trimmed();
113     if (item->text() == Lvalues[static_cast<int>(Values::null)]) {
114       return;
115     }
116     QRegExp regex(QStringLiteral("[0-9]+\\.[0-9]+\\.[0-9]+"));
117     bool isValid = regex.exactMatch(input);
118     if (isValid) {
119       QStringList digits = input.split(".");
120       QDate date = QDate(digits[2].toInt(), digits[1].toInt(), digits
      [0].toInt());
121       QDate today = QDate::currentDate();
122       if (date.isValid() and (date <= today)) {
123         item->setText(date.toString("dd.MM.yyyy"));
124         Pages[item->row()]->setDOB(QDate::fromString(item->text(), "dd.
      MM.yyyy"));
125         return;
126       }
127     }
128     else {
129       item->setText(Lvalues[static_cast<int>(Values::null)]);
130       Pages[item->row()]->setDOB(QDate());
131       QMessageBox::warning(Table->parentWidget(), Lerrors[static_cast<
      int>(Errors::incorrect_data)], Lerrors[static_cast<int>(Errors::
      incorrect_date)]);
132     }
133     break;
134   }
135   case 5:
136   {
137     QString input = item->text().trimmed();
138     if (item->text() == Lvalues[static_cast<int>(Values::null)]) {
139       return;
140     }
141     QRegExp regex(QStringLiteral("[a-zA-Z0-9\.]+[ ]?@[ ]?[a-zA-Z0-9\.]+"
      ));
142     bool isValid = regex.exactMatch(input);
143     if (isValid) {
144       item->setText(input);
145       Pages[item->row()]->setEmail(input);
146     }
147     else {
148       item->setText(Lvalues[static_cast<int>(Values::null)]);
149       Pages[item->row()]->setEmail(Lvalues[static_cast<int>(Values::null
      )]);
150       QMessageBox::warning(Table->parentWidget(), Lerrors[static_cast<
      int>(Errors::incorrect_data)], Lerrors[static_cast<int>(Errors::
      incorrect_email)]);
151     }
152     break;
153   }
```

```
154   default:
155     break;
156 
157   }
158 }
159 
160 void Book::update() {
161   Table->clearContents();
162   Table->setRowCount(0);
163   for (auto i : Pages) {
164     this->add_Page(i, false);
165   }
166 }
167 
168 void Book::clear() {
169   Pages.clear();
170   Table->clearContents();
171   Table->setRowCount(0);
172 }
173 
174 QVector<Page*> Book::getPages() {
175   return Pages;
176 }
177 
178 void Book::delete_row() {
179   QList<QTableWidgetSelectionRange> selectedRanges = Table->
     selectedRanges();
180   QSet<int> rowsToDelete;
181 
182   for (const QTableWidgetSelectionRange& range : selectedRanges) {
183     for (int row = range.topRow(); row <= range.bottomRow(); ++row) {
184       rowsToDelete.insert(row);
185     }
186   }
187   QList<int> sortedRows = rowsToDelete.toList();
188   std::sort(sortedRows.begin(), sortedRows.end(), std::greater<int>());
189 
190   for (int row : sortedRows) {
191     Table->removeRow(row);
192     Pages.remove(row);
193   }
194 }
195 
196 
197 void writeBookToJsonFile(Book* book, const QString& fileName) {
198   QFile file;
199   if (not fileName.endsWith(".json")) {
200     file.setFileName(fileName + ".json");
201   }
202   else {
203     file.setFileName(fileName);
204   }
205   if (!file.open(QIODevice::WriteOnly | QIODevice::Text)) {
206     QMessageBox::warning(book->get_Table()->parentWidget(), Lerrors[
     static_cast<int>(Errors::error_save)], Lerrors[static_cast<int>(
     Errors::table_not_saved)]);
```

```
207     return;
208   }
209 
210   QJsonArray jsonArray;
211 
212   for (auto page : book->getPages()) {
213     QJsonObject jsonObject;
214     jsonObject[Lheaders[static_cast<int>(Headers::Surname)]] = page->
      getSurname();
215     jsonObject[Lheaders[static_cast<int>(Headers::Name)]] = page->
      getName();
216     jsonObject[Lheaders[static_cast<int>(Headers::Patronymic)]] = page->
      getPatronymic();
217     jsonObject[Lheaders[static_cast<int>(Headers::Address)]] = page->
      getAddress();
218     jsonObject[Lheaders[static_cast<int>(Headers::DOB)]] = page->getDOB
      ().toString("dd.MM.yyyy");
219     jsonObject[Lheaders[static_cast<int>(Headers::Email)]] = page->
      getEmail();
220 
221     QJsonArray phoneNumbersArray = QJsonArray::fromStringList(
      QStringList(page->getPhone_Numbers().toList()));
222     QJsonArray phoneTypesArray = QJsonArray::fromStringList(page->
      getPhone_types_str().toList());
223 
224     jsonObject["Phone_Numbers"] = phoneNumbersArray;
225     jsonObject["Phone_types"] = phoneTypesArray;
226 
227     jsonArray.append(jsonObject);
228   }
229 
230   QJsonDocument jsonDoc(jsonArray);
231   file.write(jsonDoc.toJson());
232   file.close();
233 }
234 
235 bool readBookFromJsonFile(Book* book, const QString& fileName) {
236   QFile file(fileName);
237 
238   if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
239     QMessageBox::warning(book->get_Table()->parentWidget(), Lerrors[
      static_cast<int>(Errors::error_open)], Lerrors[static_cast<int>(
      Errors::not_open)]);
240     return false;
241   }
242 
243   QByteArray fileData = file.readAll();
244   file.close();
245 
246   QJsonDocument jsonDoc = QJsonDocument::fromJson(fileData);
247 
248   if (!jsonDoc.isArray()) {
249     QMessageBox::warning(book->get_Table()->parentWidget(), Lerrors[
      static_cast<int>(Errors::error_open)], Lerrors[static_cast<int>(
      Errors::bad_json)]);
250     return false;
```

```
251   }
252
253   QJsonArray jsonArray = jsonDoc.array();
254
255   for (const QJsonValue& value : jsonArray) {
256     if (!value.isObject()) {
257       continue;
258     }
259     QJsonObject obj = value.toObject();
260     Page* page = new Page();
261     page->setSurname(obj[Lheaders[static_cast<int>(Headers::Surname)]].
    toString());
262     page->setName(obj[Lheaders[static_cast<int>(Headers::Name)]].
    toString());
263     page->setPatronymic(obj[Lheaders[static_cast<int>(Headers::
    Patronymic)]].toString());
264     page->setAddress(obj[Lheaders[static_cast<int>(Headers::Address)]].
    toString());
265     page->setDOB(QDate::fromString(obj[Lheaders[static_cast<int>(Headers
    ::DOB)]].toString(), "dd.MM.yyyy"));
266     page->setEmail(obj[Lheaders[static_cast<int>(Headers::Email)]].
    toString());
267
268     QJsonArray phoneNumbersArray = obj["Phone_Numbers"].toArray();
269     QVector<QString> new_phone_numbers;
270     for (const QJsonValue& phoneNumber : phoneNumbersArray) {
271       new_phone_numbers.append(phoneNumber.toString());
272     }
273     page->setPhone_Numbers(new_phone_numbers);
274
275     QJsonArray phoneTypesArray = obj["Phone_types"].toArray();
276     QVector<int> new_phone_types;
277     for (const QJsonValue& phoneType : phoneTypesArray) {
278       new_phone_types.append(phoneType.toInt());
279     }
280     page->setPhone_types(new_phone_types);
281
282     book->add_Page(page);
283   }
284   book->update();
285   return true;
286 }
287
288 QString getSaveFilePath(QWidget* parent) {
289   QString fileName = QFileDialog::getSaveFileName(
290     parent,
291     Lerrors[static_cast<int>(Errors::open_file)],
292     "",
293     "JSON Files (*.json);;All Files (*.*)"
294   );
295
296   return fileName;
297 }
298
299 QString getOpenFilePath(QWidget* parent) {
300   QString fileName = QFileDialog::getOpenFileName(
```

```
301       parent,
302       Lerrors[static_cast<int>(Errors::open_file)],
303       "",
304       "JSON Files (*.json);;All Files (*.*)"
305     );
306 
307     return fileName;
308 }
309 
310 void Book::confirmExit(QApplication* app, QString Name) {
311     QMessageBox::StandardButton reply;
312     reply = QMessageBox::question(this->Table->parentWidget(), Lerrors[
        static_cast<int>(Errors::accept_exit)], Lerrors[static_cast<int>(
        Errors::save_exit)],
313       QMessageBox::Yes | QMessageBox::No | QMessageBox::Abort);
314 
315     if (reply == QMessageBox::Yes) {
316       writeBookToJsonFile(this, Name);
317       app->quit();
318     }
319     if (reply == QMessageBox::No) {
320       app->quit();
321     }
322     if (reply == QMessageBox::Abort) {
323       return;
324     }
325 }
326 
327 void setWidgetsEnabled(QGridLayout* layout) {
328     for (int i = 0; i < layout->rowCount(); ++i) {
329       for (int j = 0; j < layout->columnCount(); ++j) {
330         QWidget* widget = layout->itemAt(i * layout->columnCount() + j)->
        widget();
331         if (widget) {
332           widget->setEnabled(widget->isEnabled() ? false : true);
333         }
334       }
335     }
336 }
337 
338 void set_numbers_const(QTableWidget* Table) {
339     for (int i = 0; i < Table->rowCount(); i++) {
340       setWidgetsEnabled(qobject_cast<QGridLayout*>(Table->itemAt(i,
        Lheaders.size() - 1)->tableWidget()->layout()));
341     }
342 }
343 
344 void disableWidgetsInLastColumn(QTableWidget* table) {
345     QWidget* layout_widget1;
346     QWidget* layout_widget2;
347     QWidget* layout_widget3;
348     QGridLayout* grid_layout;
349     for (int i = 0; i < table->rowCount(); i++) {
350       grid_layout = qobject_cast<QGridLayout*>(table->cellWidget(i, table
        ->columnCount() - 1)->layout());
351       for (int j = 0; j < grid_layout->rowCount() - 1; j++) {
```

```
352       layout_widget1 = grid_layout->itemAtPosition(j, 0)->widget();
353       layout_widget2 = grid_layout->itemAtPosition(j, 1)->widget();
354 
355       layout_widget1->setEnabled(not layout_widget1->isEnabled());
356       layout_widget2->setEnabled(not layout_widget2->isEnabled());
357     }
358     layout_widget3 = grid_layout->itemAtPosition(grid_layout->rowCount()
      - 1, 0)->widget();
359     layout_widget3->setEnabled(not layout_widget3->isEnabled());
360   }
361 }
362 
363 QTableWidget* search(Book* book, int index_of_combo, const QString&
     request) {
364   if (book->get_Table() == nullptr || index_of_combo < 0 ||
     index_of_combo >= book->get_Table()->columnCount()) {
365     return nullptr;
366   }
367   Book* result_book = new Book(book->get_Table()->parentWidget());
368 
369   book->get_Table()->setGeometry(QRect(0, 31, 1200, 770));
370   book->get_Table()->setHorizontalHeaderLabels(Lheaders);
371   for (Page* p : book->getPages()) {
372     bool flag = false;
373     switch (index_of_combo)
374     {
375     case 0:
376       flag = p->getSurname().contains(request);
377       break;
378     case 1:
379       flag = p->getName().contains(request);
380       break;
381     case 2:
382       flag = p->getPatronymic().contains(request);
383       break;
384     case 3:
385       flag = p->getAddress().contains(request);
386       break;
387     case 4:
388       flag = p->getDOB().toString("dd.MM.yyyy").contains(request);
389       break;
390     case 5:
391       flag = p->getEmail().contains(request);
392       break;
393     case 6:
394       for (QString number : p->getPhone_Numbers()) {
395         flag = number.contains(request);
396         if (flag) {
397           break;
398         }
399       }
400       break;
401     default:
402       break;
403     }
404     if (flag) {
```

```
405         result_book->add_Page(p);
406       }
407     }
408     return result_book->get_Table();
409
410 }
```

Приложение В. Исходный код функции main

```
#include "Book.h"

#include <QWidget>
#include <QPushButton>
#include <QTableWidget>
#include <QTableView>
#include <QApplication>
#include <QHBoxLayout>
#include <QMenuBar>
#include <QLabel>
#include <QStatusBar>
#include <QWidgetAction>
#include <QMainWindow>
#include <QToolBar>
#include "Header.h"

QString TableName = QString(QStringLiteral("Новая таблица"));

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QMainWindow* w = new QMainWindow;
    QMenuBar* Menu = new QMenuBar(w);
    QToolBar* Tools = new QToolBar(w);
    Book* book = new Book(w);
    QTableWidget* Table = book->get_Table();
    QTableWidget* search_Table = new QTableWidget();

    w->setWindowTitle(QStringLiteral("Телефонный справочник: Новая таблица"))
   ;
    w->setFixedWidth(1200);
    w->setFixedHeight(800);
    Table->setGeometry(QRect(0, 31, 1200, 770));

    QMenu* menu_file = new QMenu(QStringLiteral("Файл"));
    QMenu* menu_table = new QMenu(QStringLiteral("Таблица"));

    QAction* action_file_open_file = new QAction(QStringLiteral("Открыть"
   ), w);
    menu_file->addAction(action_file_open_file);
    QAction* action_file_add_to_file = new QAction(QStringLiteral("
   Добавить к текущей таблице"), w);
    menu_file->addAction(action_file_add_to_file);
    QAction* action_file_save_file = new QAction(QStringLiteral("
   Сохранить"), w);
    menu_file->addAction(action_file_save_file);
    QAction* action_file_save_file_as = new QAction(QStringLiteral("
   Сохранить как"), w);
    menu_file->addAction(action_file_save_file_as);
    QAction* action_file_quit = new QAction(QStringLiteral("Выход"), w);
    menu_file->addAction(action_file_quit);

    QAction* action_table_new_row = new QAction(QStringLiteral("Новая
   строка"), w);
```

```
49      menu_table->addAction(action_table_new_row);
50      QAction* action_table_delete_row = new QAction(QStringLiteral("
    Удалить выделенную строку"), w);
51      menu_table->addAction(action_table_delete_row);
52      QAction* action_table_delete_all = new QAction(QStringLiteral("
    Очистить"), w);
53      menu_table->addAction(action_table_delete_all);
54      QAction* action_table_sort = new QAction(QStringLiteral("Переключить
    сортировку: Выключено"), w);
55      menu_table->addAction(action_table_sort);
56      QAction* action_table_const = new QAction(QStringLiteral("Разрешить
    редактирование: Включено"), w);
57      menu_table->addAction(action_table_const);
58
59      QObject::connect(action_table_new_row, &QAction::triggered, [book,
    Table]() {
60          if (Table->editTriggers() == QAbstractItemView::NoEditTriggers)
    {
61              QMessageBox::warning(Table->parentWidget(), QStringLiteral("
    Ошибка редактирования"),
62                  QStringLiteral("Выключен режим редактирования. Для добавления
    новых строк включите режим редактирования."));
63          }
64          else {
65              book->add_Row();
66          }
67
68          });
69      QObject::connect(action_file_save_file, &QAction::triggered, [book,
    w]() {
70          writeBookToJsonFile(book, ::TableName);
71          });
72      QObject::connect(action_file_save_file_as, &QAction::triggered, [
    book]() {
73          writeBookToJsonFile(book, getSaveFilePath());
74          });
75      QObject::connect(action_file_open_file, &QAction::triggered, [book,
    w]() {
76          QString filename = getOpenFilePath();
77          book->clear();
78          readBookFromJsonFile(book, filename);
79          ::TableName = filename.split("/").last().split(".").first();
80          w->setWindowTitle((QString(QStringLiteral("Телефонный справочник: "
    )) + TableName));
81          });
82      QObject::connect(action_file_add_to_file, &QAction::triggered, [book
    , w]() {
83          QString filename = getOpenFilePath();
84          readBookFromJsonFile(book, filename);
85          });
86
87      QObject::connect(action_table_delete_all, &QAction::triggered, book,
     &Book::clear);
88      QObject::connect(action_table_delete_row, &QAction::triggered, book,
     &Book::delete_row);
89      QObject::connect(action_file_quit, &QAction::triggered, [book, &app
```

```
    ]() {
90          book->confirmExit(&app, ::TableName);
91          });
92      QObject::connect(action_table_sort, &QAction::triggered, [Table,
    action_table_sort]() {
93          Table->setSortingEnabled(Table->isSortingEnabled() ? false :
    true);
94          action_table_sort->setText(Table->isSortingEnabled() ?
    QStringLiteral("Переключить сортировку: Выключено") : QStringLiteral("
    Переключить сортировку: Включено"));
95          });
96      QObject::connect(action_table_const, &QAction::triggered, [Table,
    action_table_const]() {
97          Table->setEditTriggers(Table->editTriggers() ==
    QAbstractItemView::NoEditTriggers ? QAbstractItemView::DoubleClicked
    : QAbstractItemView::NoEditTriggers);
98          action_table_const->setText(Table->editTriggers() ==
    QAbstractItemView::NoEditTriggers ? QStringLiteral("Разрешить
    редактирование: Выключено") : QStringLiteral("Разрешить редактирвоание:
    Включено"));
99          disableWidgetsInLastColumn(Table);
100         });
101
102     QWidget* layout_widget = new QWidget();
103     QHBoxLayout* h_layout = new QHBoxLayout();
104     QLineEdit* line_to_search = new QLineEdit();
105     QComboBox* set_column = new QComboBox();
106     QPushButton* search_button = new QPushButton(QStringLiteral("Поиск"))
    ;
107     QPushButton* cancel_button = new QPushButton(QStringLiteral("Отмена")
    );
108
109     QObject::connect(search_button, &QPushButton::clicked, [&]() {
110         search_Table = search(book, set_column->currentIndex(),
    line_to_search->text());
111         Table->hide();
112         search_Table->show();
113         search_Table->setGeometry(QRect(0, 31, 1200, 770));
114         });
115
116     QObject::connect(cancel_button, &QPushButton::clicked, [&]() {
117         search_Table->clear();
118         search_Table->hide();
119         Table->show();
120         });
121
122     set_column->addItems(Lheaders);
123     set_column->setFixedWidth(142);
124     line_to_search->setFixedWidth(160);
125     search_button->setFixedWidth(80);
126     cancel_button->setFixedWidth(80);
127
128     h_layout->addWidget(set_column);
129     h_layout->addWidget(line_to_search);
130     h_layout->addWidget(search_button);
131     h_layout->addWidget(cancel_button);
```

```
132     layout_widget->setLayout(h_layout);
133     h_layout->setContentsMargins(10, 0, 10, 0);
134 
135     Tools->setGeometry(680, 0, 520, 31);
136     Tools->setMovable(false);
137     Menu->setGeometry(0, 0, 140, 31);
138     QFont f = Menu->font();
139     f.setPointSize(11);
140     Menu->setFont(f);
141 
142     Menu->addMenu(menu_file);
143     Menu->addMenu(menu_table);
144     Tools->addWidget(layout_widget);
145     w->show();
146     return app.exec();
147 }
```