

МИНОБРНАУКИ РОССИИ

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 ПО КУРСОВОЙ РАБОТЕ ПО ДИСЦИПЛИНЕ  
ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ:**

Реализация телефонного справочника на  
языке C++ с использованием библиотеки Qt

Студент: \_\_\_\_\_ Санько В. В.

Руководитель: \_\_\_\_\_ Глазунов В. В.

«\_\_\_\_\_» \_\_\_\_\_ 20\_\_\_\_г.

Санкт-Петербург

2024

# Содержание

<b>Введение</b>	<b>3</b>
<b>1 Математическое описание</b>	<b>4</b>
1.1 Описание контактной структуры . . . . .	4
1.2 Валидация полей контакта . . . . .	4
1.3 Добавление нового контакта . . . . .	4
1.4 Поиск контактов . . . . .	5
1.5 Сохранение и загрузка данных . . . . .	5
1.6 Обработка ошибок . . . . .	5
<b>2 Особенности реализации</b>	<b>6</b>
2.1 Структуры данных Contact . . . . .	6
2.2 Таблица контактов . . . . .	6
2.3 Переменные . . . . .	6
2.4 Конструктор: MainWindow::MainWindow(QWidget *parent) . . . . .	6
2.5 Функция добавления контакта: MainWindow::on_addButton_clicked() . . . . .	7
2.6 Функция валидации контакта: MainWindow::validateContact() . . . . .	8
2.7 Функция парсинга номеров телефонов: parsePhoneNumbers() . . . . .	10
2.8 Функция добавления контакта в таблицу: addContactToTable() . . . . .	10
2.9 Функция удаления контакта: MainWindow::on_deleteButton_clicked() . . . . .	11
2.10 Функция сохранения данных в файл: MainWindow::on_saveButton_clicked() . . . . .	12
2.11 Функция сохранения данных в файл: saveToFile() . . . . .	12
2.12 Функция загрузки данных из файла: MainWindow::on_loadButton_clicked() . . . . .	14
2.13 Функция загрузки данных из файла: loadFromFile() . . . . .	14
2.14 Функция поиска по таблице: on_searchEditTextChanged() . . . . .	16
2.15 Функция обработки изменений в таблице: on_tableWidget_itemChanged() . . . . .	18
2.16 Функция сортировки по столбцам: on_tableWidget_headerClicked() . . . . .	21
<b>3 Результаты работы</b>	<b>22</b>
<b>Заключение</b>	<b>25</b>

## **Введение**

Суть работы заключается в разработке программы для управления контактами, которая обеспечивает хранение, обработку и управление данными пользователей в удобном интерфейсе. Программа позволяет добавлять, редактировать, сортировать, удалять, сохранять и импортировать контакты, включая фамилию, имя, отчество, адрес, дату рождения, адрес электронной почты и номера телефонов. Особое внимание уделено работе с регулярными выражениями, которые обеспечивают надежную защиту на ввод только корректных данных.

# 1 Математическое описание

## 1.1 Описание контактной структуры

Каждый контакт представляется как кортеж из следующих полей:

$$\text{Контакт } c = \{f, l, m, a, b, e, P\},$$

где:

- $f$  — имя (First Name),
- $l$  — фамилия (Last Name),
- $m$  — отчество (Middle Name),
- $a$  — адрес,
- $b$  — дата рождения,
- $e$  — электронная почта,
- $P = \{p_1, p_2, \dots, p_n\}$  — набор телефонных номеров.

## 1.2 Валидация полей контакта

1. Поля  $f, l, m$  считаются корректными, если удовлетворяют следующему регулярному выражению:

$$R_{\text{name}} = \wedge [A-ZА-ЯЁ] [A-Za-zA-Яа-яёЁ -]*[a-za-яё] .$$

2. Поле  $e$  корректно, если соответствует следующему регулярному выражению:

$$R_{\text{email}} = \wedge [a-zA-Z0-9._-]+ @ [a-zA-Z0-9.-]+. [a-zA-Z]\{2,\} .$$

3. Каждый номер  $p_i \in P$  считается корректным, если он удовлетворяет следующему регулярному выражению:

$$R_{\text{phone}} = \wedge \backslash +? \backslash s?\{1,3\}[-]?(\backslash (\backslash d + \backslash ))?[\backslash s-]? \backslash d + ([\backslash s-]? \backslash d+) .$$

## 1.3 Добавление нового контакта

Добавление нового контакта  $c$  в множество  $C$  происходит, если все поля контакта проходят проверку валидации. Алгоритм:

1. Считать данные контакта.
2. Проверить поля  $f, l, m, e, P$  на соответствие их регулярным выражениям.
3. Если валидация успешна, добавить  $c$  в множество  $C$ :

$$C := C \cup \{c\}.$$

## 1.4 Поиск контактов

Поиск осуществляется на основе ключевой строки  $t$ , которая сравнивается с текстовыми полями контактов. Для каждого контакта  $c$  из множества  $C$  проверяется следующее условие:

$$t \cap (f \cup l \cup m \cup a \cup e \cup P) \neq \emptyset.$$

Результатом работы алгоритма является подмножество  $C_t \subseteq C$ , для которого найдено непустое пересечение с ключевой строкой  $t$ . Результатом работы алгоритма является подмножество  $C_t \subseteq C$ , удовлетворяющее условию поиска. Все найденные совпадения в текстовых полях выделяются цветом для удобства визуального восприятия.

## 1.5 Сохранение и загрузка данных

Контакты из множества  $C$  сохраняются в текстовый файл в формате CSV, где каждая строка описывает контакт:

$$\text{line}_i = f, l, m, a, b, e, p_1; p_2; \dots; p_n.$$

При загрузке файла программа парсит каждую строку, извлекает поля и проверяет их на корректность. Корректные записи добавляются в множество  $C$ :

$$C := C \cup \{c \mid \text{валидация}(c) = \text{true}\}.$$

## 1.6 Обработка ошибок

Ошибки валидации и некорректные строки при загрузке записываются в отчёт  $E$  :

$$E = (\text{множество пар}).$$

## 2 Особенности реализации

### 2.1 Структуры данных Contact

Структура, представляющая контакт. Содержит следующие поля:

- `firstName`: Имя.
- `lastName`: Фамилия.
- `middleName`: Отчество.
- `address`: Адрес.
- `birthDate`: Дата рождения.
- `email`: Электронная почта.
- `phoneNumbers`: Список номеров телефонов.

### 2.2 Таблица контактов

Таблица отображает все контакты в виде строк. Каждая строка представляет один контакт, с данными, такими как ФИО, дата рождения, телефон и email.

### 2.3 Переменные

- `QList<Contact> contacts`: Список всех контактов.
- `QVector<Contact> filteredContacts`: Список контактов, отфильтрованных по запросу поиска.
- `bool isImporting`: Флаг, показывающий, что данные в данный момент импортируются.
- `QString errorslist`: Стока, содержащая перечень всех ошибок.

### 2.4 Конструктор: `MainWindow::MainWindow(QWidget *parent)`

**Описание:** Конструктор окна главного приложения. Инициализирует все элементы интерфейса, настраивает таблицу для отображения контактов и связывает сигналы с соответствующими слотами.

**Вход:** Принимает объект родительского виджета `QWidget`.

**Выход:** Инициализирует поля для ввода, кнопки, таблицу, соединяет сигналы и слоты.

```

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    QLineEdit *searchEdit = new QLineEdit(this);
    searchEdit->setPlaceholderText("Поиск...");
    ui->verticalLayout->addWidget(searchEdit);
    ui->birthDateEdit->setMaximumDate(QDate::currentDate());

    ui->tableWidget->setColumnWidth(6, 120);

    connect(ui->tableWidget, &QTableWidget::itemChanged, this,
&MainWindow::on_tableWidget_itemChanged);
    connect(searchEdit, &QLineEdit::textChanged, this,
&MainWindow::on_searchEdit_textChanged);
    connect(ui->tableWidget->horizontalHeader(), &QHeaderView::sectionClicked, this,
&MainWindow::on_tableWidget_headerClicked);
}

MainWindow::~MainWindow() {
    delete ui;
}

```

## 2.5 Функция добавления контакта: MainWindow::on\_addButton\_clicked()

**Описание:** Обрабатывает событие нажатия на кнопку "Добавить". Проверяет введенные данные, добавляет контакт в список и отображает его в таблице. В закомментированном фрагменте, логика очистки полей после добавления в таблицу.

**Вход:** Данные, введенные пользователем (ФИО, адрес, дата рождения, телефон, email).

**Выход:** Если данные корректны, контакт добавляется в таблицу и в список. В случае ошибки выводится сообщение об ошибке.

```

void MainWindow::on_addButton_clicked() {
    Contact contact;
    contact.firstName = ui->firstNameEdit->text().trimmed();

```

```

contact.lastName = ui->lastNameEdit->text().trimmed();
contact.middleName = ui->middleNameEdit->text().trimmed();
contact.address = ui->addressEdit->text().trimmed();
contact.birthDate = ui->birthDateEdit->date();
contact.email = ui->emailEdit->text().trimmed();
//contact.phoneNumbers = ui->phoneEdit->text().split(", ", QString::SkipEmptyParts);

QString phoneInput = ui->phoneEdit->text();
contact.phoneNumbers = parsePhoneNumbers(phoneInput);

if (contact.phoneNumbers.isEmpty()) {
    QMessageBox::warning(this, "Ошибка ввода", "Все номера телефонов некорректны!");
    return;
}

/*
ui->firstNameEdit->clear();
ui->lastNameEdit->clear();
ui->middleNameEdit->clear();
ui->addressEdit->clear();
ui->birthDateEdit->setDate(QDate::currentDate());
ui->emailEdit->clear();
ui->phoneEdit->clear();
*/
if (validateContact(contact)) {
    contacts.append(contact);
    addContactToTable(contact);
}
}

```

## 2.6 Функция валидации контакта: MainWindow::validateContact()

**Описание:** Проверяет корректность введенных данных (имя, телефон, email) перед добавлением контакта. Ниже приведены функции проверки имени и email. На вход в первую функцию подается имя, фамилия или отчество и получаем true только в том случае, если строка начинается с буквы верхнего регистра, затем следуют буквы нижнего регистра, после может идти пробел или дефис и после обязательно должны быть буквы. Незначащие пробелы удаляются.

Предусмотрена буква Ё для строк, использующих кирилицу.

Корректность почты проверяется также посредством регулярного выражение и должно удовлетворять условию: начинается с букв нижнего и верхнего регистров или цифр, допустимо использование точки и нижнего подчеркивания, затем следует символ @, за ним буквы, цифры и допустимые знаки, после обязательно точка и после точки, буквы, при чем их количество должно быть не меньше двух. И так если контакт прошел этап проверки ФИО и почты, функция проверяет контейнер номеров, если он пуст функция выдаст false, иначе контакт прошел проверку и будет использоваться дальше. В случае возникновения ошибки, функция определяет в каком или каких полях были допущены ошибки и выведет диалоговое окно с указанием поля, содержащего ошибку.

**Вход:** Объект Contact, содержащий все данные, введенные пользователем.

**Выход:** Возвращает true, если все данные корректны, и false, если хотя бы одно поле неверно.

```
bool MainWindow::validateContact(const Contact &contact) {  
    if (!isValidName(contact.firstName) || !isValidName(contact.lastName)  
        || !isValidName(contact.middleName)) {  
        QMessageBox::warning(this, "Ошибка", "Имя, фамилия или отчество некорректны!");  
        return false;  
    }  
  
    if (!isValidEmail(contact.email)) {  
        QMessageBox::warning(this, "Ошибка", "Некорректный email!");  
        return false;  
    }  
  
    if (contact.phoneNumbers.isEmpty()) {  
        QMessageBox::warning(this, "Ошибка", "Все номера телефонов некорректны!");  
        return false;  
    }  
  
    return true;  
}  
  
bool MainWindow::isValidName(const QString &name) {
```

```

QRegularExpression regex(R"(^[A-ZА-ЯЁ] [А-За-зА-Яа-яёЁ\-\ ]*[а-за-яё]$)");
return regex.match(name.trimmed()).hasMatch();
}

bool MainWindow::isValidEmail(const QString &email) {
    QRegularExpression regex(R"(^[a-zA-Z0-9\.\_]+\@[a-zA-Z0-9\.\_]+\.[a-zA-Z]{2,}\$)");
    return regex.match(email.trimmed()).hasMatch();
}

```

## 2.7 Функция парсинга номеров телефонов: parsePhoneNumbers()

**Описание:** Разбирает строку с номерами телефонов, очищает её от лишних символов и приводит к стандартному виду.

**Вход:** Стока с номерами телефонов, разделенными запятыми.

**Выход:** Список корректных номеров телефонов.

```

QStringList parsePhoneNumbers(const QString &input) {
    QStringList numbers = input.split(", ", QString::SkipEmptyParts);
    QStringList validNumbers;

    QRegularExpression phoneRegex(R"^(\\+?\\s?\\d{1,3}[\\s-]?((\\(\\d+\\))?[\\s-]?\\d+([\\s-]?\\d+)*$))");

    for (const QString &number : numbers) {
        QString trimmedNumber = number.trimmed();
        if (phoneRegex.match(trimmedNumber).hasMatch()) {
            QString sanitizedNumber = trimmedNumber.remove(QRegularExpression(R"([^\d]+)"));
            validNumbers.append(sanitizedNumber);
        } else {
            qDebug() << "Неверный номер:" << trimmedNumber;
        }
    }
    return validNumbers;
}

```

## 2.8 Функция добавления контакта в таблицу: addContactToTable()

**Описание:** Добавляет контакт в таблицу для отображения, путем добавления каждого поля контакта в соответствующую колонку таблицы. Номера добавляются каждый на новую строку,

предусмотрена логика расширения ячейки и всего ряда в соответствии с количеством номеров. В случае успешно выполненной операции, выводиться окно с сообщением о добавлении контакта.

**Вход:** Объект `Contact`, который нужно отобразить в таблице.

**Выход:** Контакт отображается в таблице в виде новой строки.

```
void MainWindow::addContactToTable(const Contact &contact) {  
    int row = ui->tableView->rowCount();  
    ui->tableView->insertRow(row);  
  
    ui->tableView->setItem(row, 0, new QTableWidgetItem(contact.firstName));  
    ui->tableView->setItem(row, 1, new QTableWidgetItem(contact.lastName));  
    ui->tableView->setItem(row, 2, new QTableWidgetItem(contact.middleName));  
    ui->tableView->setItem(row, 3, new QTableWidgetItem(contact.address));  
    ui->tableView->setItem(row, 4,  
        new QTableWidgetItem(contact.birthDate.toString("dd.MM.yyyy")));  
    ui->tableView->setItem(row, 5, new QTableWidgetItem(contact.email));  
  
    QTableWidgetItem *phoneItem = new QTableWidgetItem(contact.phoneNumbers.join(", "));  
    phoneItem->setFlags(phoneItem->flags() | Qt::ItemIsEditable);  
    ui->tableView->setItem(row, 6, phoneItem);  
  
    phoneItem->setTextAlignment(Qt::AlignLeft | Qt::AlignVCenter);  
    phoneItem->setText(contact.phoneNumbers.join("\n"));  
  
    ui->tableView->resizeRowToContents(row);  
  
    if(!isImporting){  
        QMessageBox::information(this, "Успех", "Контакт добавлен.");  
    }  
}
```

## 2.9 Функция удаления контакта: `MainWindow::on_deleteButton_clicked()`

**Описание:** Удаляет выбранный контакт из таблицы и списка. Удаление осуществляется по выбранной ячейке или строке.

**Вход:** Выбранная строка или ячейка таблицы, представляющая контакт.

**Выход:** Контакт удаляется из таблицы и списка контактов.

```

void MainWindow::on_deleteButton_clicked() {
    int row = ui->tableView->currentRow();
    if (row >= 0) {
        ui->tableView->removeRow(row);
        contacts.removeAt(row);
    }
}

```

## 2.10 Функция сохранения данных в файл: MainWindow::on\_saveButton\_clicked()

**Описание:** Обрабатывает сигнал о сохранении данных контакта в файл CSV.

**Вход:** Путь до файла, куда нужно сохранить данные.

**Выход:** Вызов функции, которая сохранит файл.

```

void MainWindow::on_saveButton_clicked() {
    QString filePath = QFileDialog::getSaveFileName(this, "Сохранить файл", "", "CSV (*.csv)");
    if (!filePath.isEmpty()) {
        saveToFile(filePath);
    }
}

```

## 2.11 Функция сохранения данных в файл: saveToFile()

**Описание:** Записывает все данные контактов в файл CSV. В первую очередь происходит проверка на возможность открыть файл. Далее данные из ячеек заносятся в свои переменные, номера приводят к подходящему для хранения формату, удаляются пробелы и дефисы. Также номера записываются в строку, в которой они разделены символом точкой с запятой. затем в файл заносятся данные и файл закрывается.

**Вход:** Путь к файлу, куда нужно сохранить данные.

**Выход:** Данные всех контактов сохраняются в файл CSV.

```

void MainWindow::saveToFile(const QString &filePath) {
    QFile file(filePath);
    if (!file.open(QIODevice::WriteOnly | QIODevice::Text)) {
        QMessageBox::warning(this, "Ошибка", "Не удалось открыть файл для записи!");
        return;
    }
    QTextStream out(&file);

```

```

for (int i = 0; i < ui->tableWidget->rowCount(); ++i) {
    if (!ui->tableWidget->isRowHidden(i)) {
        QString firstName = ui->tableWidget->item(i, 0)->text();
        QString lastName = ui->tableWidget->item(i, 1)->text();
        QString middleName = ui->tableWidget->item(i, 2)->text();
        QString address = ui->tableWidget->item(i, 3)->text();
        QString birthDate = ui->tableWidget->item(i, 4)->text();
        QString email = ui->tableWidget->item(i, 5)->text();
        QString phoneNumbersString = ui->tableWidget->item(i, 6)->text();

        QStringList phoneNumbers = phoneNumbersString.split(";");
        QStringList sanitizedPhoneNumbers;
        for (QString &number : phoneNumbers) {
            number = number.trimmed()
                .remove(QRegularExpression(R"([^\d+\s-]+)"))
                .replace(" ", "")
                .replace("-", "");
            sanitizedPhoneNumbers.append(number);
        }

        QString sanitizedPhoneNumbersString = sanitizedPhoneNumbers.join(";");
        out << firstName << ","
            << lastName << ","
            << middleName << ","
            << address << ","
            << birthDate << ","
            << email << ","
            << sanitizedPhoneNumbersString << "\n";
    }
}

file.close();
}

```

## 2.12 Функция загрузки данных из файла: MainWindow::on\_loadButton\_clicked()

**Описание:** Обрабатывает сигнал о загрузке данных контакта в файл CSV.

**Вход:** Путь до CSV файла для загрузки данных.

**Выход:** Вызов функции, которая загрузит файл.

```
void MainWindow::on_loadButton_clicked() {  
    QString filePath = QFileDialog::getOpenFileName(this, "Открыть файл", "", "CSV (*.csv)");  
    if (!filePath.isEmpty()) {  
        loadFromFile(filePath);  
    }  
}
```

## 2.13 Функция загрузки данных из файла: loadFromFile()

**Описание:** Первоочередно устанавливается флаг о том, что файл импортируется. Затем проверка на открытие файла и сразу после все данные таблицы очищаются. Этую логику можно исправить в зависимости от пожеланий, то-есть можно загружать контакты к тем, что уже в таблице или же очищать таблицу перед добавлением. Файл построчно читается, производит проверку на количество ячеек в строке и записывает данные из таблицы в контакт. Затем контакт добавляется. Это повторяется до тех пор, пока не прочитаем все строки файла. В конце выводится окно с полной информацией о количестве добавленных контактов и всех ошибках вызванных при добавлении. Флаг импортируется снова возвращается к значению false, что означает что импортирование окончено.

**Вход:** Путь к CSV файлу для загрузки данных.

**Выход:** Данные из CSV файла загружаются в список контактов и отображаются в таблице.

```
void MainWindow::loadFromFile(const QString &filePath) {  
    isImporting = true;  
  
    QFile file(filePath);  
  
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {  
        QMessageBox::warning(this, "Ошибка", "Не удалось открыть файл.");  
        isImporting = false;  
        return;  
    }
```

```

contacts.clear();

ui->tableView->setRowCount(0);

QTextStream in(&file);
QString report;
int addedCount = 0;
int lineNumber = 0;

while (!in.atEnd()) {
    lineNumber++;
    QString line = in.readLine();
    QStringList parts = line.split(",");
    if (parts.size() < 7) {
        QMessageBox::warning(this, "Ошибка", "Неполное количество данных в строке.");
        continue;
    }

    Contact contact;
    contact.firstName = parts[0];
    contact.lastName = parts[1];
    contact.middleName = parts[2];
    contact.address = parts[3];
    contact.birthDate = QDate::fromString(parts[4], "dd.MM.yyyy");
    contact.email = parts[5];
    contact.phoneNumbers = parts[6].split(":");

    contacts.append(contact);
    addContactToTable(contact);
    addedCount++;

}

report = QString("Импорт завершён.\nДобавлено контактов: %1\n\n").arg(addedCount)
+ report + errorslist;
QMessageBox::information(this, "Импорт завершён.", report.trimmed());
errorslist = "";

```

```
    isImporting = false;  
}
```

## 2.14 Функция поиска по таблице: on\_searchEditTextChanged()

**Описание:** В строку ввода записывается строка, далее функция проверяет все строки и все столбцы таблицы. Предусмотрена проверка на соответствие всех элементов, чтобы визуально не перегружать таблицу, в случае совпадения всех ячеек выделение текста выполняться не будет, в противном случае выделится текст всех ячеек, где было найдено совпадение. Затем еще раз программа проходится уже по выделенной таблице и отыскивает строки, где нет ни одной выделенной ячейки. Поиск осуществляется при помощи флага rowHasMatch. Такие строки скрываются при помощи функции setRowHidden(). Повторная проверка на совпадение всех элементов. И в конце все отфильтрованные контакты сохраняются, это сделано для того, чтобы можно было реализовать дальнейшую логику работы с отфильтрованными контактами.

Следует также рассказать о вспомогательной функции resetHighlighting(), она проходит по всем элементам и меняет цвет текста на стандартный.

Цвет текста меняется на зеленый, так как он позволяет заметно выделить текст и при этом оставить его читаемым, альтернативным решением было закраска фона в желтый. При желании сменить логику, она частично закомментирована.

**Вход:** Текст, введенный в поле поиска.

**Выход:** Фильтруются и отображаются только те строки, которые соответствуют запросу.

```
void MainWindow::on_searchEditTextChanged(const QString &text) {  
    filteredContacts.clear();  
    int matchCount = 0;  
    int totalCount = ui->tableWidget->rowCount() * ui->tableWidget->columnCount();  
  
    if (text.isEmpty()) {  
        resetHighlighting();  
    }  
  
    for (int i = 0; i < ui->tableWidget->rowCount(); ++i) {  
        for (int j = 0; j < ui->tableWidget->columnCount(); ++j) {  
            QTableWidgetItem *item = ui->tableWidget->item(i, j);  
            if (item && item->text().contains(text, Qt::CaseInsensitive)) {  
                ++matchCount;  
            }  
        }  
    }  
}
```

```

        if (matchCount < totalCount) {

            item->setTextColor(Qt::green);

        }

    } else {

        if (item) {

            item->setTextColor(Qt::black);

        }

    }

}

for (int i = 0; i < ui->tableWidget->rowCount(); ++i) {

    bool rowHasMatch = false;

    for (int j = 0; j < ui->tableWidget->columnCount(); ++j) {

        if (ui->tableWidget->item(i, j)->textColor() == Qt::green) {

            rowHasMatch = true;

            break;

        }

    }

    ui->tableWidget->setRowHidden(i, !rowHasMatch);

}

if (matchCount == totalCount) {

    resetHighlighting();

}

if (matchCount < totalCount) {

    for (int i = 0; i < ui->tableWidget->rowCount(); ++i) {

        bool matches = false;

        for (int j = 0; j < ui->tableWidget->columnCount(); ++j) {

            if (ui->tableWidget->item(i, j)->textColor() == Qt::green) {

                matches = true;

                break;

            }

        }

    }

}

```

```

        if (matches) {
            Contact contact;
            contact.firstName = ui->tableView->item(i, 0)->text();
            contact.lastName = ui->tableView->item(i, 1)->text();
            contact.middleName = ui->tableView->item(i, 2)->text();
            contact.address = ui->tableView->item(i, 3)->text();
            contact.birthDate = QDate::fromString(ui->tableView->item(i, 4)->text(),
                "dd.MM.yyyy");
            contact.email = ui->tableView->item(i, 5)->text();
            contact.phoneNumbers = ui->tableView->item(i, 6)->text().split(";");
            filteredContacts.append(contact);
        }
    }
}

void MainWindow::resetHighlighting() {
    for (int i = 0; i < ui->tableView->rowCount(); ++i) {
        for (int j = 0; j < ui->tableView->columnCount(); ++j) {
            QTableWidgetItem *item = ui->tableView->item(i, j);
            if (item) {
                item->setTextColor(Qt::black);
            }
        }
    }
}

```

## 2.15 Функция обработки изменений в таблице: on\_tableWidget\_itemChanged()

**Описание:** Обрабатывает изменения в ячейках таблицы. При помощи конструкции if else находится номер ячейки который редактируют, происходит проверка на загрузку файла. При загрузке файла в случае многократных ошибок будет огромное количество диалоговых окон, как раз для этого используется флаг импортирования, чтобы функция редактирования посыпала диалоговые окна об ошибках только при редактировании. В случае импортации ошибки просто записываются в строку и один сообщением отображают перечень проблем. Благодаря тому, что

логика редактирования определяет конкретную ячейку, можно выводить сообщение об ошибке конкретного поля. Ранее при заполнении контакта, если пользователь допустил ошибку в ФИО, программа не могла выяснить в каком именно из этих 3 полей была допущена ошибка, так как для них всех использовалась единая функция проверки, однако это функция может точно определить и вывести пользователю сообщение в чем он ошибся. При возникновении ошибки, поле вернется в исходный вариант, если же редактирование было корректным, поле заполниться новым значением.

**Вход:** Измененные данные в ячейке таблицы.

**Выход:** Если данные корректны, они обновляются в таблице. В случае ошибок выводится предупреждение.

```
void MainWindow::on_tableWidget_itemChanged(QTableWidgetItem *item) {  
    int row = item->row();  
    int column = item->column();  
  
    Contact &contact = contacts[row];  
  
    if (column == 0) {  
        QString firstName = item->text().trimmed();  
        if (!isValidName(firstName)) {  
            if(isImporting){  
                errorslist += "Некорректное имя!\n";  
            }  
            else{  
                QMessageBox::warning(this, "Ошибка", "Некорректное имя!");  
            }  
            item->setText(contact.firstName);  
        }  
        return;  
    }  
    contact.firstName = firstName;  
}  
else if (column == 1) {  
    QString lastName = item->text().trimmed();  
    if (!isValidName(lastName)) {  
        if(isImporting){  
            errorslist += "Некорректная фамилия!\n";  
        }  
        else{  
    }
```

```

    QMessageBox::warning(this, "Ошибка", "Некорректная фамилия!");

}

item->setText(contact.lastName);

return;
}

contact.lastName = lastName;

} else if (column == 2) {

QString middleName = item->text().trimmed();

if (!isValidName(middleName)) {

if(isImporting){

errorslist += "Некорректное отчество!\n";

}

else{

QMessageBox::warning(this, "Ошибка", "Некорректное отчество!");

}

item->setText(contact.middleName);

return;
}

contact.middleName = middleName;

} else if (column == 3) {

contact.address = item->text().trimmed();

} else if (column == 4) { // Birth Date

QDate birthDate = QDate::fromString(item->text(), "dd.MM.yyyy");

if (!birthDate.isValid()) {

if(isImporting){

errorslist += "Некорректная дата рождения!\n";

}

else{

QMessageBox::warning(this, "Ошибка", "Некорректная дата рождения!");

}

item->setText(contact.birthDate.toString("dd.MM.yyyy"));

return;
}

contact.birthDate = birthDate;

} else if (column == 5) {

QString email = item->text().trimmed();

```

```

    if (!isValidEmail(email)) {
        if(isImporting){
            errorslist += "Некорректный email!\n";
        }
        else{
            QMessageBox::warning(this, "Ошибка", "Некорректный email!");
        }
        item->setText(contact.email);
        return;
    }
    contact.email = email;
} else if (column == 6) { // Phone Numbers
    QString phoneNumbersText = item->text().trimmed();
    QStringList phoneNumbers = parsePhoneNumbers(phoneNumbersText);
    if (phoneNumbers.isEmpty()) {
        if(isImporting){
            errorslist += "Некорректный номер телефона!\n";
        }
        else{
            //QMessageBox::warning(this, "Ошибка", "Некорректный номер телефона!");
        }
        item->setText(contact.phoneNumbers.join(", "));
        return;
    }
    contact.phoneNumbers = phoneNumbers;
}
}

```

## 2.16 Функция сортировки по столбцам: on\_tableWidget\_headerClicked()

**Описание:** Сортирует таблицу по выбранному столбцу.

**Вход:** Номер столбца, по которому нужно произвести сортировку.

**Выход:** Отсортированная таблица.

```

void MainWindow::on_tableWidget_headerClicked(int column){
    ui->tableWidget->sortByColumn(column, Qt::AscendingOrder);
}

```

### 3 Результаты работы

В ходе разработки данной программы был создан графический интерфейс для управления контактами с использованием фреймворка Qt. Приложение позволяет пользователю добавлять, редактировать, сортировать, удалять и сохранять контакты в формате CSV, а также выполнять поиск по введённым данным. Для ввода и проверки информации реализованы механизмы валидации, такие как проверка корректности имени, адреса электронной почты и номеров телефонов. Также предусмотрены функции импорта и экспорта данных с обработкой ошибок формата.

Дополнительно была добавлена функциональность для динамического фильтрации и подсветки данных при поиске. Удобство работы с программой достигается благодаря автоматическому форматированию данных, подсказкам при ошибках ввода и пользовательскому уведомлению о результатах действий.

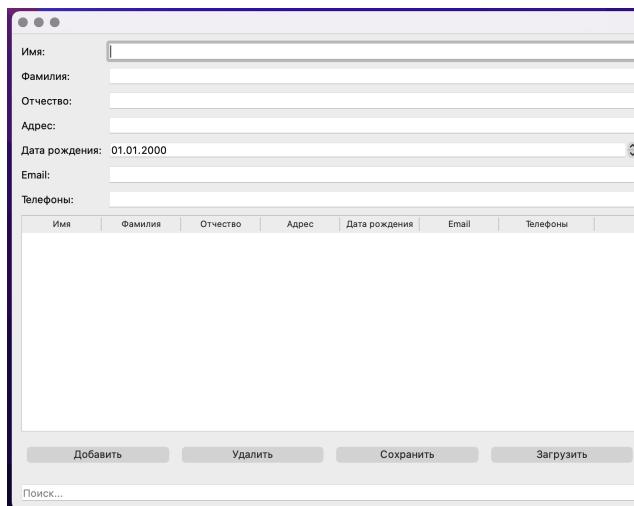


Рис. 1: Вывод консоли

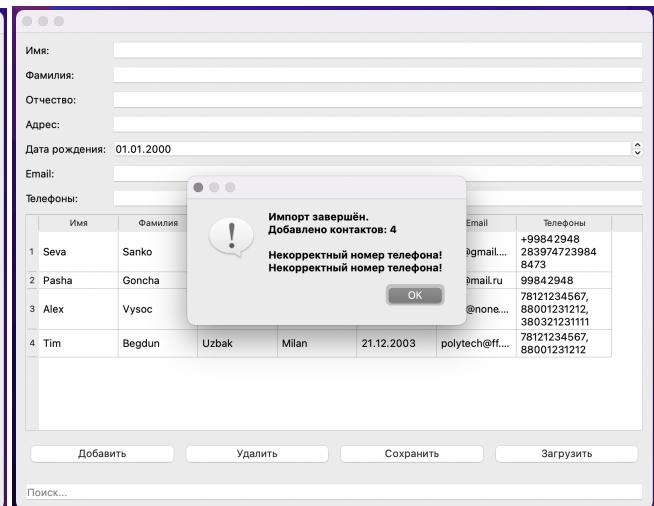


Рис. 2: Импортированный файл

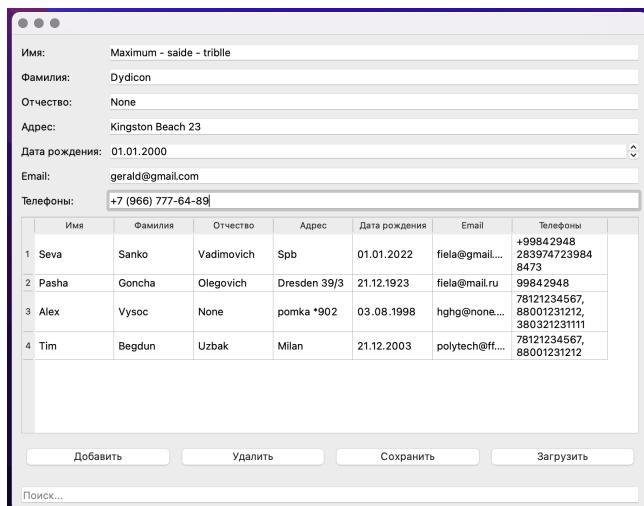


Рис. 3: Добавление контакта

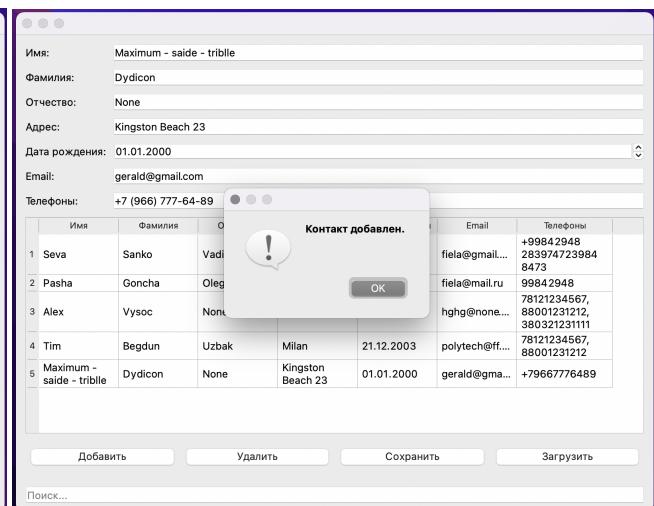


Рис. 4: Контакт добавлен

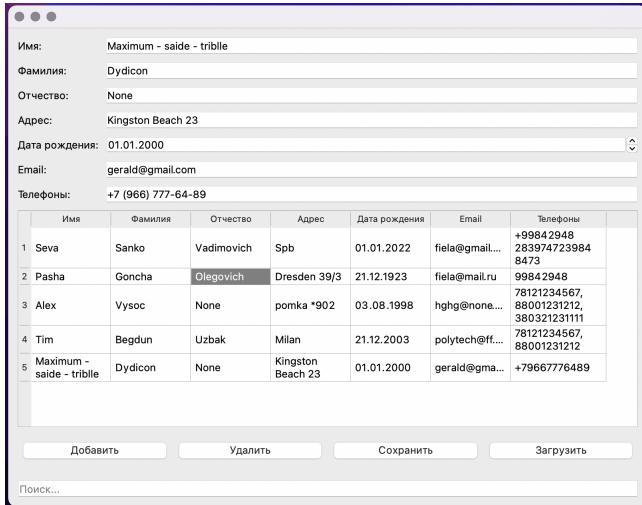


Рис. 5: Выбор ячейки

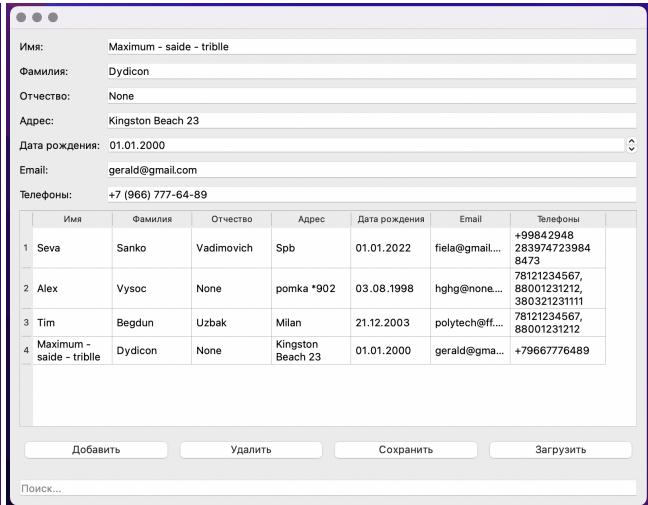


Рис. 6: Удаление контакта по выбранной ячейке

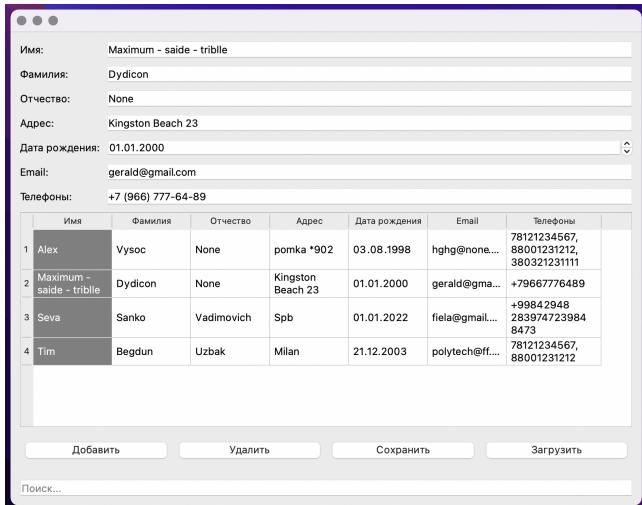


Рис. 7: Пример сортировки

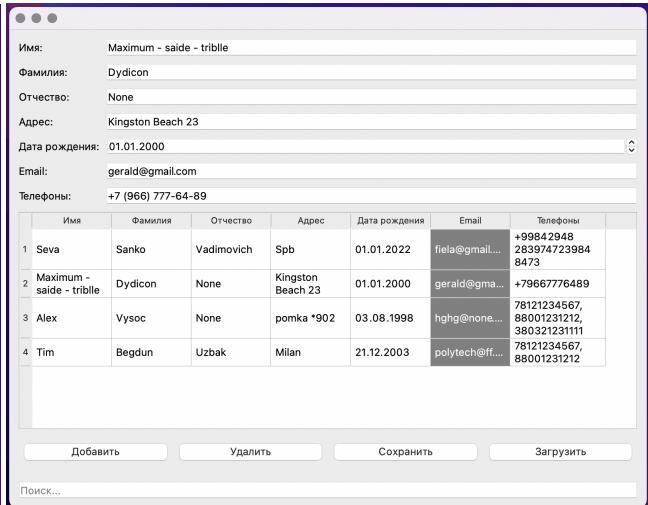


Рис. 8: Пример сортировки

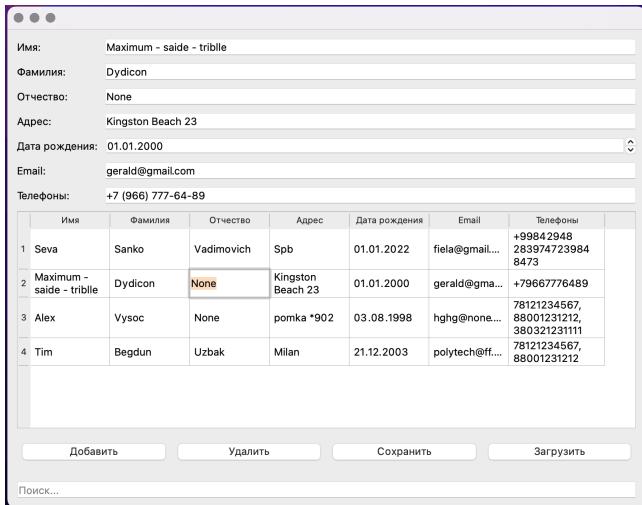


Рис. 9: Редактирование ячейки

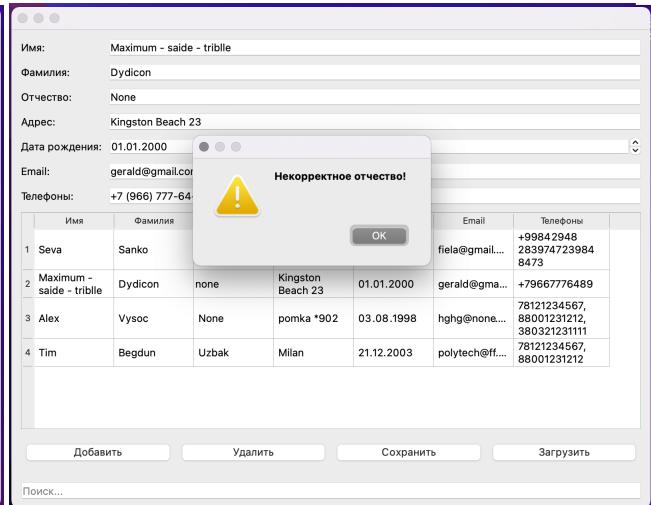


Рис. 10: Некорректное редактирование

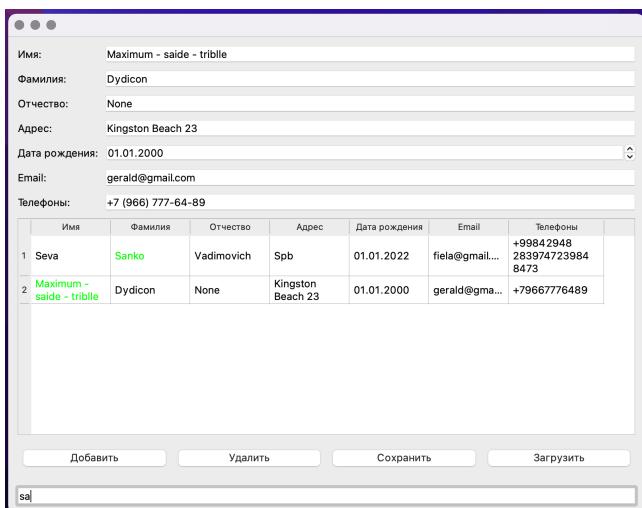


Рис. 11: Поиск

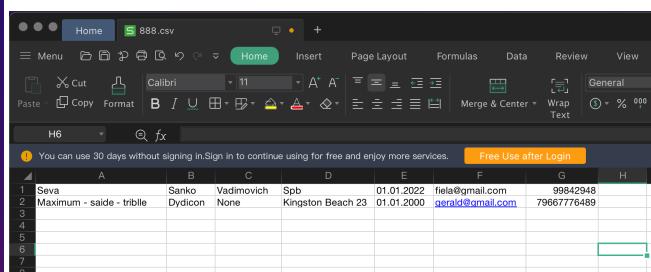


Рис. 12: Хранение отфильтрованных контактов

## **Заключение**

В курсовой работе были выполнены все поставленные задачи. В ходе выполнения работы удалось углубиться в понимание и особенности класса QTable.

Главными достоинствами программы можно назвать масштабируемость и гибкость программных интерфейсов. Её легко дополнить другими методами для более удобной работы со справочником.

Главным недостатком программы является её внешний вид, который требует улучшений для более удобного последующего использования. Также частичное дублирование кода и отсутствие поддержки современных форматов данных.

## Приложение

### Исходный код Main.cpp

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

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

### Исходный код MainWindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QList>
#include <QDate>
#include <QString>
#include <QRegularExpression>
#include <QTableWidgetItem>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

struct Contact {
    QString firstName;
    QString lastName;
    QString middleName;
    QString address;
    QDate birthDate;
```

```

    QString email;
    QStringList phoneNumbers;
};

class MainWindow : public QMainWindow {
Q_OBJECT

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

private slots:
    void on_addButton_clicked();
    void on_deleteButton_clicked();
    void on_saveButton_clicked();
    void on_loadButton_clicked();
    void on_searchEditTextChanged(const QString &text);
    void on_tableWidget_itemChanged(QTableWidgetItem *item);
    void on_tableWidget_headerClicked(int column);

private:
    Ui::MainWindow *ui;
    QList<Contact> contacts;
    bool isImporting = false;
    QString errorslist;
    QVector<Contact> filteredContacts;

    void addContactToTable(const Contact &contact);
    bool validateContact(const Contact &contact);
    void saveToFile(const QString &filePath);
    void loadFromFile(const QString &filePath);
    bool isValidEmail(const QString &email);
    //bool isValidPhone(const QString &phone);
    bool isValidName(const QString &name);
    void resetHighlighting();
}

```

```
};

#endif // MAINWINDOW_H
```

## Исходный код MainWindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QMessageBox>
#include <QFile>
#include <QTextStream>
#include <QRegularExpression>
#include <QFileDialog>
#include <QDebug>
#include <QTimer>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    QLineEdit *searchEdit = new QLineEdit(this);
    searchEdit->setPlaceholderText("Поиск...");
    ui->verticalLayout->addWidget(searchEdit);
    ui->birthDateEdit->setMaximumDate(QDate::currentDate());

    ui->tableView->setColumnWidth(6, 120);

    connect(ui->tableView, &QTableWidget::itemChanged, this,
            &MainWindow::on_tableWidget_itemChanged);
    connect(searchEdit, &QLineEdit::textChanged, this, &MainWindow::on_searchEditTextChanged);
    connect(ui->tableView->horizontalHeader(), &QHeaderView::sectionClicked, this,
            &MainWindow::on_tableWidget_headerClicked);
}
```

```

MainWindow::~MainWindow() {
    delete ui;
}

void MainWindow::on_tableWidget_headerClicked(int column){
    ui->tableWidget->sortByColumn(column, Qt::AscendingOrder);
}

QStringList parsePhoneNumbers(const QString &input) {
    QStringList numbers = input.split(", ", QString::SkipEmptyParts);
    QStringList validNumbers;

    QRegularExpression phoneRegex(R"((^+?\s?\d{1,3}[\s-]?(\\(\d+\\))?[\\s-]?\d+([\s-]?\d+)*$))");

    for (const QString &number : numbers) {
        QString trimmedNumber = number.trimmed();
        if (phoneRegex.match(trimmedNumber).hasMatch()) {
            QString sanitizedNumber = trimmedNumber.remove(QRegularExpression(R"([^\d]+)"));
            validNumbers.append(sanitizedNumber);
        } else {
            qDebug() << "Неверный номер:" << trimmedNumber;
        }
    }
    return validNumbers;
}

void MainWindow::addContactToTable(const Contact &contact) {
    int row = ui->tableWidget->rowCount();
    ui->tableWidget->insertRow(row);

    ui->tableWidget->setItem(row, 0, new QTableWidgetItem(contact.firstName));
    ui->tableWidget->setItem(row, 1, new QTableWidgetItem(contact.lastName));
    ui->tableWidget->setItem(row, 2, new QTableWidgetItem(contact.middleName));
    ui->tableWidget->setItem(row, 3, new QTableWidgetItem(contact.address));
    ui->tableWidget->setItem(row, 4,
        new QTableWidgetItem(contact.birthDate.toString("dd.MM.yyyy")));
}

```

```

ui->tableView->setItem(row, 5, new QTableWidgetItem(contact.email));

QTableWidgetItem *phoneItem = new QTableWidgetItem(contact.phoneNumbers.join(", "));
phoneItem->setFlags(phoneItem->flags() | Qt::ItemIsEditable);
ui->tableView->setItem(row, 6, phoneItem);

phoneItem->setTextAlignment(Qt::AlignLeft | Qt::AlignVCenter);
phoneItem->setText(contact.phoneNumbers.join("\n"));

ui->tableView->resizeRowToContents(row);

if(!isImporting){
    QMessageBox::information(this, "Успех", "Контакт добавлен.");
}

}

bool MainWindow::validateContact(const Contact &contact) {
    if (!isValidName(contact.firstName) ||
        !isValidName(contact.lastName) || !isValidName(contact.middleName)) {
        QMessageBox::warning(this, "Ошибка", "Имя, фамилия или отчество некорректны!");
        return false;
    }

    if (!isValidEmail(contact.email)) {
        QMessageBox::warning(this, "Ошибка", "Некорректный email!");
        return false;
    }

    if (contact.phoneNumbers.isEmpty()) {
        QMessageBox::warning(this, "Ошибка", "Все номера телефонов некорректны!");
        return false;
    }
    return true;
}

bool MainWindow::isValidName(const QString &name) {

```

```

QRegularExpression regex(R"(^[A-ZА-ЯЁ] [А-За-зА-Яа-яёЁ\-\ ]*[а-за-яё]$/)");
return regex.match(name.trimmed()).hasMatch();
}

bool MainWindow::isValidEmail(const QString &email) {
    QRegularExpression regex(R"(^[a-zA-Z0-9\.\_]+\@[a-zA-Z0-9\.\_]+\.[a-zA-Z]{2,}\$)");
    return regex.match(email.trimmed()).hasMatch();
}

void MainWindow::on_addButton_clicked() {
    Contact contact;

    contact.firstName = ui->firstNameEdit->text().trimmed();
    contact.lastName = ui->lastNameEdit->text().trimmed();
    contact.middleName = ui->middleNameEdit->text().trimmed();
    contact.address = ui->addressEdit->text().trimmed();
    contact.birthDate = ui->birthDateEdit->date();
    contact.email = ui->emailEdit->text().trimmed();

    QString phoneInput = ui->phoneEdit->text();
    contact.phoneNumbers = parsePhoneNumbers(phoneInput);

    if (contact.phoneNumbers.isEmpty()) {
        QMessageBox::warning(this, "Ошибка ввода", "Все номера телефонов некорректны!");
        return;
    }
    /*
    ui->firstNameEdit->clear();
    ui->lastNameEdit->clear();
    ui->middleNameEdit->clear();
    ui->addressEdit->clear();
    ui->birthDateEdit->setDate(QDate::currentDate());
    ui->emailEdit->clear();
    ui->phoneEdit->clear();
    */
    if (validateContact(contact)) {
        contacts.append(contact);
    }
}

```

```

        addContactToTable(contact);
    }

}

void MainWindow::on_deleteButton_clicked() {
    int row = ui->tableView->currentRow();
    if (row >= 0) {
        ui->tableView->removeRow(row);
        contacts.removeAt(row);
    }
}

void MainWindow::on_saveButton_clicked() {
    QString filePath = QFileDialog::getSaveFileName(this, "Сохранить файл", "", "CSV (*.csv)");
    if (!filePath.isEmpty()) {
        saveToFile(filePath);
    }
}

void MainWindow::on_loadButton_clicked() {
    QString filePath = QFileDialog::getOpenFileName(this, "Открыть файл", "", "CSV (*.csv)");
    if (!filePath.isEmpty()) {
        loadFromFile(filePath);
    }
}

void MainWindow::saveToFile(const QString &filePath) {
    QFile file(filePath);
    if (!file.open(QIODevice::WriteOnly | QIODevice::Text)) {
        QMessageBox::warning(this, "Ошибка", "Не удалось открыть файл для записи!");
        return;
    }
    QTextStream out(&file);

    for (int i = 0; i < ui->tableView->rowCount(); ++i) {
        if (!ui->tableView->isRowHidden(i)) {

```

```

QString firstName = ui->tableWidget->item(i, 0)->text();
QString lastName = ui->tableWidget->item(i, 1)->text();
QString middleName = ui->tableWidget->item(i, 2)->text();
QString address = ui->tableWidget->item(i, 3)->text();
QString birthDate = ui->tableWidget->item(i, 4)->text();
QString email = ui->tableWidget->item(i, 5)->text();
QString phoneNumbersString = ui->tableWidget->item(i, 6)->text();

QStringList phoneNumbers = phoneNumbersString.split(";");
QStringList sanitizedPhoneNumbers;
for (QString &number : phoneNumbers) {
    number = number.trimmed()
        .remove(QRegularExpression(R"([^\d+\s-])"))
        .replace(" ", "")
        .replace("-", "");
    sanitizedPhoneNumbers.append(number);
}

QString sanitizedPhoneNumbersString = sanitizedPhoneNumbers.join(":");

out << firstName << ","
    << lastName << ","
    << middleName << ","
    << address << ","
    << birthDate << ","
    << email << ","
    << sanitizedPhoneNumbersString << "\n";
}

file.close();
}

void MainWindow::loadFromFile(const QString &filePath) {
    isImporting = true;
    QFile file(filePath);

```

```

if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
    QMessageBox::warning(this, "Ошибка", "Не удалось открыть файл.");
    isImporting = false;
    return;
}

contacts.clear();
ui->tableWidget->setRowCount(0);

QTextStream in(&file);
QString report;
int addedCount = 0;
int lineNumber = 0;

while (!in.atEnd()) {
    lineNumber++;
    QString line = in.readLine();
    QStringList parts = line.split(",");
    if (parts.size() < 7) {
        QMessageBox::warning(this, "Ошибка", "Неполное количество данных в строке.");
        continue;
    }

    Contact contact;
    contact.firstName = parts[0];
    contact.lastName = parts[1];
    contact.middleName = parts[2];
    contact.address = parts[3];
    contact.birthDate = QDate::fromString(parts[4], "dd.MM.yyyy");
    contact.email = parts[5];
    contact.phoneNumbers = parts[6].split(";");

    contacts.append(contact);
    addContactToTable(contact);
    addedCount++;
}

```

```

report = QString("Импорт завершён.\nДобавлено контактов: %1\n\n").arg(addedCount) + report
+ errorslist;

QMessageBox::information(this, "Импорт завершён.", report.trimmed());
errorslist = "";
isImporting = false;
}

void MainWindow::on_searchEditTextChanged(const QString &text) {
    filteredContacts.clear();
    int matchCount = 0;
    int totalCount = ui->tableWidget->rowCount() * ui->tableWidget->columnCount();

    if (text.isEmpty()) {
        resetHighlighting();
    }

    for (int i = 0; i < ui->tableWidget->rowCount(); ++i) {
        for (int j = 0; j < ui->tableWidget->columnCount(); ++j) {
            QTableWidgetItem *item = ui->tableWidget->item(i, j);
            if (item && item->text().contains(text, Qt::CaseInsensitive)) {
                ++matchCount;

                if (matchCount < totalCount) {
                    item->setTextColor(Qt::green);
                }
            } else {
                if (item) {
                    item->setTextColor(Qt::black);
                }
            }
        }
    }
}

for (int i = 0; i < ui->tableWidget->rowCount(); ++i) {
    bool rowHasMatch = false;
    for (int j = 0; j < ui->tableWidget->columnCount(); ++j) {

```

```

        if (ui->tableWidget->item(i, j)->textColor() == Qt::green) {
            rowHasMatch = true;
            break;
        }
    }

    ui->tableWidget->setRowHidden(i, !rowHasMatch);
}

if (matchCount == totalCount) {
    resetHighlighting();
}

if (matchCount < totalCount) {
    for (int i = 0; i < ui->tableWidget->rowCount(); ++i) {
        bool matches = false;
        for (int j = 0; j < ui->tableWidget->columnCount(); ++j) {
            if (ui->tableWidget->item(i, j)->textColor() == Qt::green) {
                matches = true;
                break;
            }
        }
        if (matches) {
            Contact contact;
            contact.firstName = ui->tableWidget->item(i, 0)->text();
            contact.lastName = ui->tableWidget->item(i, 1)->text();
            contact.middleName = ui->tableWidget->item(i, 2)->text();
            contact.address = ui->tableWidget->item(i, 3)->text();
            contact.birthDate = QDate::fromString(ui->tableWidget->item(i, 4)->text(),
"dd.MM.yyyy");
            contact.email = ui->tableWidget->item(i, 5)->text();
            contact.phoneNumbers = ui->tableWidget->item(i, 6)->text().split(";");
            filteredContacts.append(contact);
        }
    }
}

```

```

    }

}

void MainWindow::resetHighlighting() {
    for (int i = 0; i < ui->tableWidget->rowCount(); ++i) {
        for (int j = 0; j < ui->tableWidget->columnCount(); ++j) {
            QTableWidgetItem *item = ui->tableWidget->item(i, j);
            if (item) {
                item->setTextColor(Qt::black);
            }
        }
    }
}

void MainWindow::on_tableWidget_itemChanged(QTableWidgetItem *item) {
    int row = item->row();
    int column = item->column();
    Contact &contact = contacts[row];

    if (column == 0) {
        QString firstName = item->text().trimmed();
        if (!isValidName(firstName)) {
            if(isImporting){
                errorslist += "Некорректное имя!\n";
            }
            else{
                QMessageBox::warning(this, "Ошибка", "Некорректное имя!");
            }
            item->setText(contact.firstName);
            return;
        }
        contact.firstName = firstName;
    } else if (column == 1) {
        QString lastName = item->text().trimmed();
        if (!isValidName(lastName)) {
            if(isImporting){

```

```

        errorslist += "Некорректная фамилия!\n";
    }
    else{
        QMessageBox::warning(this, "Ошибка", "Некорректная фамилия!");
    }
    item->setText(contact.lastName);
    return;
}

contact.lastName = lastName;
} else if (column == 2) {

    QString middleName = item->text().trimmed();
    if (!isValidName(middleName)) {
        if(isImporting){
            errorslist += "Некорректное отчество!\n";
        }
        else{
            QMessageBox::warning(this, "Ошибка", "Некорректное отчество!");
        }
        item->setText(contact.middleName);
        return;
    }

    contact.middleName = middleName;
} else if (column == 3) {

    contact.address = item->text().trimmed();
} else if (column == 4) { // Birth Date

    QDate birthDate = QDate::fromString(item->text(), "dd.MM.yyyy");
    if (!birthDate.isValid()) {
        if(isImporting){
            errorslist += "Некорректная дата рождения!\n";
        }
        else{
            QMessageBox::warning(this, "Ошибка", "Некорректная дата рождения!");
        }
        item->setText(contact.birthDate.toString("dd.MM.yyyy"));
        return;
    }
}

```

```

        contact.birthDate = birthDate;
    } else if (column == 5) {
        QString email = item->text().trimmed();
        if (!isValidEmail(email)) {
            if(isImporting){
                errorslist += "Некорректный email!\n";
            }
            else{
                QMessageBox::warning(this, "Ошибка", "Некорректный email!");
            }
            item->setText(contact.email);
            return;
        }
        contact.email = email;
    } else if (column == 6) { // Phone Numbers
        QString phoneNumbersText = item->text().trimmed();
        QStringList phoneNumbers = parsePhoneNumbers(phoneNumbersText);
        if (phoneNumbers.isEmpty()) {
            if(isImporting){
                errorslist += "Некорректный номер телефона!\n";
            }
            else{
                //QMessageBox::warning(this, "Ошибка", "Некорректный номер телефона!");
            }
            item->setText(contact.phoneNumbers.join(", "));
            return;
        }
        contact.phoneNumbers = phoneNumbers;
    }
}

```