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»

Отчёт по курсовой работе

Объектно-ориентированное программирование

Создание приложения «Телефонный справочник» с использованием библиотеки Qt

Студент
группы 5130201/30002 _____ Михайлова А. А.

Преподаватель _____ Кирпиченко С. Р.

«____»_____ 2025 г.

Санкт-Петербург, 2025

Содержание

1 Постановка задачи **4**

2 Реализация **5**
2.1 Использованные классы 5
2.2 Класс MainWindow 6
2.3 Функция setupTable 9
2.4 Функция updateSalaryColumn 10
2.5 Функция setupValidators 12
2.6 Функция handleCellChanged 12
2.7 Функция validateHome 16
2.8 Функция validateStat 16
2.9 Функция validatePhoneNumber 17
2.10 Функция validateName 18
2.11 Функция validateBirthDate 18
2.12 Функция validateEmail 19
2.13 Функция removeSelectedRows 19
2.14 Функция search 20
2.15 Функция cancelSearch 22
2.16 Функция loadDataFromFile 23
2.17 Функция saveDataToFile 25
2.18 Функция saveVisibleDataToFile 26
2.19 Функция sortTableByColumn 28

3 Тестирование приложения **29**

Заключение **36**

Приложение А. Исходный код **37**
А1. Исходный код файла mainwindow.h 37
А2. Исходный код файла mainwindow.cpp 39
А3. Исходный код файла main.cpp 55

Введение

Qt Creator C++ — это мощная интегрированная среда разработки (IDE), созданная для разработки кроссплатформенных приложений на языке C++ с использованием фреймворка Qt. Qt предлагает инструменты для работы с графикой, обработки событий, взаимодействия с файлами и базами данных, а также предоставляет разнообразные виджеты, такие как таблицы и поля ввода, для удобства взаимодействия пользователя с приложением. Благодаря этому фреймворку было разработано приложение «Телефонный справочник».

Телефонный справочник — это приложение, предназначенное для эффективного хранения, редактирования и поиска контактной информации пользователей. Оно поддерживает основные операции: добавление, редактирование, удаление и поиск записей. Важным требованием является внедрение средств валидации данных, что предотвратит ввод недопустимой информации, такой как некорректные телефонные номера или неверные форматы дат.

1 Постановка задачи

Необходимо создать приложение «Телефонный справочник» с использованием фреймворка Qt, которое может хранить, сохранять в файл, загружать из файла, удалять, выполнять поиск записей с контактной информацией.

1. Реализация таблицы для представления записей пользователю с использованием класса QTableWidget.

2. Реализация поисковой строки, а также кнопок в графическом интерфейсе приложения: кнопки «Сохранить данные», «Загрузить данные», «Удалить выделенные строки», «Очистить результаты поиска».

3. Обеспечить проверку вводимых данных с использованием следующих регулярных выражений:

   (a) Фамилия, Имя или Отчество — должны содержать только буквы и цифры различных алфавитов, а также дефис и пробел, но при этом должны начинаться только на заглавные буквы, и не могли бы оканчиваться или начинаться на дефис. Все незначимые пробелы перед и после данных должны удаляться.

   (b) Телефон должен быть записан в международном формате, а храниться внутри как последовательность цифр.

   (c) Дата рождения должна быть меньше текущей даты, число месяцев в дате должно быть от 1 до 12, число дней от 1 до 31, причем должно учитываться различное число дней в месяце и високосные года.

   (d) E-mail должен содержать в себе имя пользователя состоящее из латинских букв и цифр, символ разделения пользователя и имени домена(@), а также сам домен состоящий из латинских букв и цифр. Все незначащие пробелы (включая пробелы перед и после символа @) должны быть удалены.

4. Реализация добавления и удаления записи в самописном классе MainWindow, а также редактирования всех полей.

5. Реализовать возможность сортировки отображаемых данных по указанному полю и поиск по нескольким полям.

6. Предусмотреть запись и чтение файла формата (*.csv) с использованием класса QFile, позволяя пользователю сохранять данные на диск и загружать их при последующем запуске приложения.

7. Реализация дополнительного задания: создание столбца с зарплатой, закрашивающего ячейку в зависимости от наибольшей в столбце.

2 Реализация

2.1 Использованные классы

Класс ‘QMainWindow‘ представляет собой основное окно в приложении. Он служит контейнером для размещения виджетов и управления макетом.

‘QTableWidget‘ — это виджет, который позволяет отображать данные в виде таблицы. Он поддерживает управление строками и столбцами. С помощью этого класса можно легко создавать и редактировать таблицы, предоставляя пользователям возможность взаимодействовать с данными в удобной форме.

Класс ‘QRegularExpressionValidator‘ используется для валидации пользовательского ввода, основанного на регулярных выражениях. С его помощью можно задать шаблон, которому должен соответствовать вводимый текст, что позволяет гарантировать, что данные, введенные пользователем, имеют корректный формат.

‘QMessageBox‘ предоставляет возможность отображения простых диалоговых окон для уведомлений, запросов подтверждения или сообщений об ошибках.

‘QRegularExpression‘ — это класс, который предоставляет возможности для работы с регулярными выражениями, позволяя выполнять поиск, замену и соответствие строкам.

Класс ‘QPushButton‘ представляет собой интерактивную кнопку, на которую может реагировать пользователь.

‘QDate‘ — это класс, который позволяет работать с датами, включая их создание, форматирование и вычисление разницы между ними. Он предоставляет удобные методы для проверки корректности дат и управления ими, что особенно полезно в приложениях, связанных с хранением и обработкой временной информации.

‘QVBoxLayout‘ используется для управления расположением виджетов в вертикальном направлении. Этот класс позволяет автоматически организовывать виджеты так, чтобы они располагались один под другим.

Класс ‘QWidget‘ является основным классом для всех виджетов в Qt. Он предоставляет базовую функциональность для создания и управления графическими элементами.

‘QSet‘ — это контейнер, который хранит уникальные значения и обеспечивает быстрый доступ к ним. Этот класс полезен для хранения данных, при необходимости избегая дубликатов.

‘QLineEdit‘ представляет собой однострочное текстовое поле, где пользователи могут вводить данные.

‘QFileDialog‘ — это класс, который предоставляет стандартный диалог для выбора файлов и директорий. Он позволяет пользователям выбирать, открывать и сохранять файлы при взаимодействии с приложением.

'QTextStream' - предоставляет удобные средства для чтения и записи текста. Использован для импорта и экспорта контактных данных в текстовые файлы.

'QHeaderView' - отвечает за отображение заголовков столбцов в виджетах таблиц.

2.2 Класс MainWindow

В конструкторе происходит настройка интерфейса главного окна приложения. Начинается с инициализации основного виджета и создания таблицы для отображения данных. Затем создаются различные элементы управления, такие как кнопки для поиска, удаления строк, загрузки и сохранения данных, а также кнопка для отмены поиска. При нажатии на эти кнопки связывается их функциональность с соответствующими методами, что позволяет выполнять необходимые действия.

Также создается текстовое поле для ввода поискового запроса. При изменении текста в этом поле автоматически выполняется поиск данных в таблице. Все элементы управления располагаются вертикально в центральном виджете, который устанавливается как основной элемент окна.

В функции используются следующие сигналы и слоты для взаимодействия между пользовательским интерфейсом и логикой приложения:

1. **Сигнал** `QPushButton::clicked` **для** `cancelSearchButton`

   Слот: `MainWindow::cancelSearch`

   - **Описание:** Срабатывает при нажатии кнопки "Очистить результаты поиска". Вызывает метод для сброса результатов поиска.

2. **Сигнал** `QLineEdit::textChanged` **для** `searchLineEdit`

   Слот: Лямбда-функция

   - **Описание:** Срабатывает при изменении текста в поле ввода. Вызывает метод `search` с текущим текстом для выполнения поиска.

3. **Сигнал** `QPushButton::clicked` **для** `deleteButton`

   Слот: `MainWindow::removeSelectedRows`

   - **Описание:** Срабатывает при нажатии кнопки "Удалить выделенные строки". Вызывает метод для удаления выбранных строк из таблицы.

4. **Сигнал** `QPushButton::clicked` **для** `loadButton`

   Слот: `MainWindow::loadDataFromFile`

- **Описание:** Срабатывает при нажатии кнопки "Загрузить данные". Вызывает метод для загрузки данных из CSV-файла.

5. **Сигнал** `QPushButton::clicked` **для** `saveButton`

   Слот: `MainWindow::saveDataToFile`

   - **Описание:** Срабатывает при нажатии кнопки "Сохранить данные". Вызывает метод для сохранения данных в CSV-файл.

6. **Сигнал** `QPushButton::clicked` **для** `saveVisibleButton`

   Слот: `MainWindow::saveVisibleDataToFile`

   - **Описание:** Срабатывает при нажатии кнопки "Сохранить видимые данные". Вызывает метод для сохранения только видимых данных в CSV-файл.

7. **Сигнал** `QTableWidget::cellChanged` **для** `tableWidget`

   Слот: `MainWindow::handleCellChanged`

   - **Описание:** Срабатывает при изменении содержимого ячейки таблицы. Вызывает метод для обработки изменений ячейки.

8. **Сигнал** `QHeaderView::sectionClicked` **для** `tableWidget->horizontalHeader()`

   Слот: `MainWindow::sortTableByColumn`

   - **Описание:** Срабатывает при клике на заголовок столбца таблицы. Вызывает метод для сортировки таблицы по выбранному столбцу.

Листинг 1: Конструктор MainWindow

```
MainWindow::MainWindow(QWidget *parent)
: QMainWindow(parent), tableWidget(new QTableWidget(this)), searchLineEdit(new QLineEdit(this))
{
        setupTable();
        setupValidators();
        searchLineEdit->setPlaceholderText("Введите текст для поиска...");
        connect(searchLineEdit, &QLineEdit::textChanged, this, [this]() {
                search(searchLineEdit->text());
        });
        QPushButton *deleteButton = new QPushButton("Удалить выделенные строки", this);
```

```
11        deleteButton->setStyleSheet("background-color: #E6E6FA");
12        connect(deleteButton, &QPushButton::clicked, this, &MainWindow::removeSelectedRows);
13
14        QPushButton *loadButton = new QPushButton("Загрузить данные", this);
15        loadButton->setStyleSheet("background-color: #E6E6FA");
16        connect(loadButton, &QPushButton::clicked, this, &MainWindow::loadDataFromFile);
17
18        QPushButton *saveButton = new QPushButton("Сохранить данные", this);
19        saveButton->setStyleSheet("background-color: #E6E6FA");
20        connect(saveButton, &QPushButton::clicked, this, &MainWindow::saveDataToFile);
21
22        QWidget *centralWidget = new QWidget(this);
23        QVBoxLayout *layout = new QVBoxLayout(centralWidget);
24        QPushButton *cancelSearchButton = new QPushButton("Очистить результаты поиска", this);
25        cancelSearchButton->setStyleSheet("background-color: #E6E6FA");
26        connect(cancelSearchButton, &QPushButton::clicked, this, &MainWindow::cancelSearch);
27
28        QPushButton *saveVisibleButton = new QPushButton("Сохранить видимые данные", this);
29        connect(saveVisibleButton, &QPushButton::clicked, this, &MainWindow::saveVisibleDataToFile);
30        layout->addWidget(saveVisibleButton);
31
32
33        layout->addWidget(tableWidget);
34        layout->addWidget(saveButton);
35        layout->addWidget(loadButton);
36        layout->addWidget(deleteButton);
37        setCentralWidget(centralWidget);
38        layout->addWidget(searchLineEdit);
39        searchLineEdit->setStyleSheet("background-color
```

```
    :␣#F8F8FF");
40          layout->addWidget(cancelSearchButton);
41          connect(tableWidget, &QTableWidget::cellChanged
    , this, &MainWindow::handleCellChanged);
42
43          QPalette Pal(palette());
44          Pal.setColor(QPalette::Background, QColor(176,
    196, 222));
45          this->setAutoFillBackground(true);
46          this->setPalette(Pal);
47          connect(searchLineEdit, &QLineEdit::textChanged
    , this, [this]() {
48                  search(searchLineEdit->text());
49          });
50
51          tableWidget->setSortingEnabled(false);
52          connect(tableWidget->horizontalHeader(), &
    QHeaderView::sectionClicked, this, &MainWindow::
    sortTableByColumn);
53          }
54
```

2.3 Функция setupTable

Функция настраивает таблицу, устанавливая количество строк и столбцов, а также определяя заголовки для каждого столбца. Она включает возможность сортировки данных, задает ширину столбцов. В результате таблица становится готовой для ввода и отображения контактных данных.

Листинг 2: Функция setupTable

```
1  void MainWindow::setupTable()
2  {
3  tableWidget->setRowCount(30);
4  tableWidget->setColumnCount(10);
5  QStringList headers {"Имя", "Фамилия", "Отчество", "Адрес"
       , "Дата␣рождения", "Email", "Рабочий␣номер", "
     Стационарный␣номер", "Домашний␣номер", "Зарплата"};
6  tableWidget->setHorizontalHeaderLabels(headers);
7  tableWidget->setSortingEnabled(true);
8  for (int col = 0; col < tableWidget->columnCount(); ++
     col) {
9          tableWidget->setColumnWidth(col, 165);
10 }
```

```
11 for (int row = 0; row < tableWidget->rowCount(); ++row)
       {
12         for (int col = 0; col < tableWidget->
   columnCount(); ++col) {
13                 QTableWidgetItem *item = new
   QTableWidgetItem();
14                 item->setBackground(QColor(248, 248,
   255));
15                 item->setForeground(QBrush(Qt::black));
16                 tableWidget->setItem(row, col, item);
17         }
18 }
19 }
```

2.4 Функция updateSalaryColumn

Функция обновляет цвет фона ячеек в столбце заработной платы в зависимости от значений зарплаты сотрудников. Сначала она проходит по всем строкам таблицы, чтобы найти максимальную зарплату, определяя ее среди всех введенных значений. Затем функция снова итерируется по строкам, преобразуя значения заработной платы в проценты относительно максимальной зарплаты, а затем рассчитывает цвет фона для каждой ячейки на основе этого процента. Ячейки с низкими зарплатами становятся более яркими, а с высокими — более насыщенными, что позволяет визуально выделять различия в доходах.

Листинг 3: Функция updateSalaryColumn

```
1  void MainWindow::updateSalaryColumn()
2  {
3          int maxSalary = 0;
4          for (int row = 0; row < tableWidget->rowCount()
   ; ++row) {
5                  QTableWidgetItem *salaryItem =
   tableWidget->item(row, 9);
6                  if (salaryItem) {
7                          bool ok;
8                          int salary = salaryItem->text()
   .toInt(&ok);
9                          if (ok && salary > maxSalary) {
10                                 maxSalary = salary;
11                         }
12                 }
13         }
```

```
14          for (int row = 0; row < tableWidget->rowCount()
    ; ++row) {
15                  QTableWidgetItem *salaryItem =
    tableWidget->item(row, 9);
16                  if (salaryItem) {
17                          bool ok;
18                          int salary = salaryItem->text()
    .toInt(&ok);
19                          if (ok && maxSalary > 0) {
20                                  int percentage = (
    static_cast<float>(salary) / maxSalary) * 100;
21                                  QColor fillColor =
    QColor(255, 255 * (100 - percentage) / 100, 255 *
    (100 - percentage) / 100);
22                                  salaryItem->
    setBackground(fillColor);
23                          }
24                  }
25          }
26  }
```

Блок-схема для данной функции представлена на – рис 1.

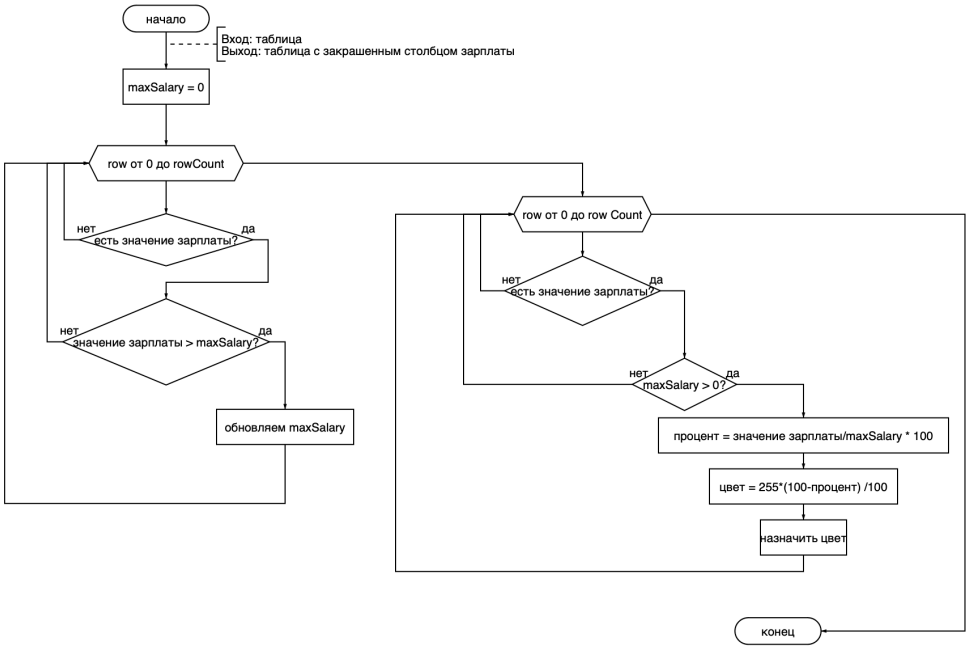

Рис. 1: Блок-схема

2.5 Функция setupValidators

Функция устанавливает валидаторы для проверки корректности ввода телефонных номеров в приложении. Каждый валидатор создается с использованием регулярного выражения, которое позволяет вводить номера, начинающиеся с ‘8‘ или ‘+7‘, за которыми следуют три цифры (код региона) и группы цифр, составляющие основной номер. Формат допускает пробелы, дефисы.

Листинг 4: Функция setupValidators

```
1 void MainWindow::setupValidators()
2 {
3         phoneValidator = new
  QRegularExpressionValidator(QRegularExpression("
  ^(8|\\+7)\\s?\\$?\\d{3}\\$?\\s?\\d{3}[\\s-]?\\d
  {2}[\\s-]?\\d{2}$"), this);
4         homePhoneValidator = new
  QRegularExpressionValidator(QRegularExpression("
  ^(8|\\+7)\\s?\\$?\\d{3}\\$?\\s?\\d{3}[\\s-]?\\d
  {2}[\\s-]?\\d{2}$"), this);
5         statPhoneValidator = new
  QRegularExpressionValidator(QRegularExpression("
  ^(8|\\+7)\\s?\\$?\\d{3}\\$?\\s?\\d{3}[\\s-]?\\d
  {2}[\\s-]?\\d{2}$"), this);
6 }
```

2.6 Функция handleCellChanged

Функция отвечает за обработку изменений в ячейках таблицы, проверяя корректность введенных данных в зависимости от столбца. Сначала она блокирует сигналы, чтобы избежать ненужных вызовов при изменении текста ячейки. Затем, если ячейка пустая, обработка прекращается. Введенный текст очищается от лишних пробелов и проверяется на соответствие требованиям:

- Для первых трех столбцов (имя, фамилия, отчество) проводится валидация с помощью функции ‘validateName()‘. Если текст не соответствует критериям (должен содержать только буквы, цифры, дефисы и пробелы, начинаться с заглавной буквы), пользователю показывается сообщение об ошибке, и ячейка очищается.

- В случае с датой рождения проверяется, что дата меньше текущей даты с помощью ‘validateBirthDate()‘. Если проверка не пройдена, отображается предупреждение, и ячейка очищается.

- Для электронной почты используется ‘validateEmail()‘, а для телефонных номеров — ‘validatePhoneNumber()‘, ‘validateHome()‘ и ‘validateStat()‘. При ошибках текст также удаляется.

- Если изменяется последний столбец (зарплата), вызывается функция ‘updateSalaryColumn()‘ для обновления информации в этом столбце.

В конце функция разблокирует сигналы, чтобы восстановить нормальную работу обработки изменений.

Листинг 5: Функция handleCellChanged

```
void MainWindow::handleCellChanged(int row, int column)
{
        tableWidget->blockSignals(true);
        if (!tableWidget->item(row, column)) {
                return;
        }
        QString text = tableWidget->item(row, column)->
  text().trimmed();
        text.replace(QRegularExpression("\\s+"), " ");
        switch (column) {
                case 0:
                if (!validateName(text)) {
                        QMessageBox::warning(this, "
  Ошибка", "Ошибка ввода. Имя должно содержать только буквы и
  цифры различных алфавитов, а также дефис и пробел, но при
  этом должно начинаться только на заглавные буквы, и не могли
  бы оканчиваться или начинаться на дефис.");
                        tableWidget->item(row, column)
  ->setText("");
                } else {
                        tableWidget->item(row, column)
  ->setText(text);
                }
                break;

                case 1:
                if (!validateName(text)) {
                        QMessageBox::warning(this, "
  Ошибка", "Ошибка ввода. Фамилия должна содержать только
  буквы и цифры различных алфавитов, а также дефис и пробел,
  но при этом должно начинаться только на заглавные буквы, и
  не могли бы оканчиваться или начинаться на дефис.");
                        tableWidget->item(row, column)
  ->setText("");
```

```
23                  } else {
24                          tableWidget->item(row, column)
    ->setText(text);
25                  }
26                  break;
27
28                  case 2:
29                  if (!validateName(text)) {
30                          QMessageBox::warning(this, "
    Ошибка", "Ошибка ввода. Отчество должно содержать только
    буквы и цифры различных алфавитов, а также дефис и пробел,
    но при этом должно начинаться только на заглавные буквы, и
    не могли бы оканчиваться или начинаться на дефис.");
31                          tableWidget->item(row, column)
    ->setText("");
32                  } else {
33                          tableWidget->item(row, column)
    ->setText(text);
34                  }
35                  break;
36
37                  case 4:
38                  if (!validateBirthDate(text)) {
39                          QMessageBox::warning(this, "
    Ошибка", "Дата рождения должна быть меньше текущей даты");
40                          tableWidget->item(row, column)
    ->setText("");
41                  } else {
42                          tableWidget->item(row, column)
    ->setText(text);
43                  }
44                  break;
45
46                  case 5:
47                  if (!validateEmail(text)) {
48                          QMessageBox::warning(this, "
    Ошибка", "E-mail должен содержать в себе имя пользователя
    состоящее из латинских букв и цифр, символ разделения
    пользователя и имени домена (@), а также сам домен состоящий
    из латинских букв и цифр. Все незначащие пробелы должны быть
    удалены.");
49                          tableWidget->item(row, column)
    ->setText("");
```

```
                } else {
                        tableWidget->item(row, column)
    ->setText(text);
                }
                break;

                case 6:
                if (!validatePhoneNumber(text)) {
                        tableWidget->item(row, column)
    ->setText("");
                } else {
                        tableWidget->item(row, column)
    ->setText(text);
                }
                break;
                case 7:
                if (!validateHome(text)) {
                        tableWidget->item(row, column)
    ->setText("");
                } else {
                        tableWidget->item(row, column)
    ->setText(text);
                }
                break;
                case 8:
                if (!validateStat(text)) {
                        tableWidget->item(row, column)
    ->setText("");
                } else {
                        tableWidget->item(row, column)
    ->setText(text);
                }
                break;
                case 9:
                if(column==9){
                        updateSalaryColumn();
                }
                default:
                break;
        }
        tableWidget->blockSignals(false);
}
```

2.7 Функция validateHome

Функция проверяет корректность ввода домашнего номера телефона. Сначала она создает копию переданного номера и удаляет все символы, кроме цифр. Затем проверяется длина очищенного номера: если он меньше 10 цифр, выводится сообщение об ошибке, предлагающее пользователю ввести корректный номер, и функция возвращает значение ‘false‘. Если длина номера корректная, очищенный номер сохраняется обратно в переменную и функция возвращает ‘true‘, указывая на успешную валидацию.

Листинг 6: Функция validateHome

```
bool MainWindow::validateHome(QString &homephoneNumber)
{
        QString cleanedPhone = homephoneNumber;
        cleanedPhone.remove(QRegularExpression("[^0-9]"
  ));
        if (cleanedPhone.length() < 10) {
                QMessageBox::warning(this, "Ошибка␣
  ввода", "Введите␣корректный␣домашний␣номер␣телефона␣шаблоны(:
  ␣8␣987␣654␣32␣10,␣+7␣900␣123␣45␣67,␣8-912-345-67-89)
  .");
                return false;
        }
        homephoneNumber = cleanedPhone;
        return true;
}
```

2.8 Функция validateStat

Функция проверяет корректность ввода стационарного номера телефона. Сначала она создает копию переданного номера и удаляет все символы, кроме цифр. Затем проверяется длина очищенного номера: если он меньше 10 цифр, выводится сообщение об ошибке, предлагающее пользователю ввести корректный номер, и функция возвращает значение ‘false‘. Если длина номера корректная, очищенный номер сохраняется обратно в переменную и функция возвращает ‘true‘, указывая на успешную валидацию.

Листинг 7: Функция validateStat

```
bool MainWindow::validateStat(QString &statphoneNumber)
{
        QString cleanedPhone = statphoneNumber;
        cleanedPhone.remove(QRegularExpression("[^0-9]"
  ));
```

```
5           if (cleanedPhone.length() < 10) {
6                   QMessageBox::warning(this, "Ошибка ввода", "Введите корректный стационарный номер телефона шаблоны(: 8 987 654 32 10, +7 900 123 45 67, 8-912-345-67-89).");
7                   return false;
8           }
9           statphoneNumber = cleanedPhone;
10          return true;
11  }
```

2.9 Функция validatePhoneNumber

Функция проверяет корректность ввода номера телефона. Сначала она создает копию переданного номера и удаляет все символы, кроме цифр. Затем проверяется длина очищенного номера: если он меньше 10 цифр, выводится сообщение об ошибке, предлагающее пользователю ввести корректный номер, и функция возвращает значение ‘false‘. Если длина номера корректная, очищенный номер сохраняется обратно в переменную и функция возвращает ‘true‘, указывая на успешную валидацию.

Листинг 8: Функция validatePhoneNumber

```
1   bool MainWindow::validatePhoneNumber(QString &phoneNumber)
2   {
3           QString cleanedPhone = phoneNumber;
4           cleanedPhone.remove(QRegularExpression("[^0-9]"));
5           if (cleanedPhone.length() < 10) {
6                   QMessageBox::warning(this, "Ошибка ввода", "Введите корректный рабочий номер телефона шаблоны(: 8 987 654 32 10, +7 900 123 45 67, 8-912-345-67-89).");
7                   return false;
8           }
9           phoneNumber = cleanedPhone;
10          return true;
11  }
```

2.10 Функция validateName

Функция проверяет правильность введенного имени с помощью регулярного выражения. Она гарантирует, что имя начинается с заглавной буквы (латиницы или кириллицы), за которой могут следовать буквы, цифры и пробелы или дефисы в середине. Имя также должно заканчиваться на букву или цифру. Если введенное имя соответствует этим критериям, функция возвращает ‘true‘, что означает успешную валидацию, и ‘false‘ в противном случае.

Листинг 9: Функция validateName

```
bool MainWindow::validateName(const QString &name)
{
        QRegularExpression regex("^[A-AZЯЁ-][A-Za-Az
  Яаяё--0-9]*(?:[␣-][A-Za-AzЯаяё--0-9]+)*[A-Za-Az
  Яаяё--0-9]$");
        return regex.match(name).hasMatch();
}
```

2.11 Функция validateBirthDate

Функция проверяет корректность ввода даты рождения. Сначала она преобразует строку с датой в объект ‘QDate‘, используя формат "дд.мм.гггг". Затем она сравнивает полученную дату с текущей, чтобы убедиться, что дата рождения является валидной и предшествует сегодняшнему дню. Также проверяется, что месяц и день находятся в допустимых диапазонах для месяца (1-12 для месяца и 1 до количества дней в месяце). Если все условия выполняются, функция возвращает ‘true‘, что свидетельствует о правильном вводе даты, иначе — ‘false‘, указывая на ошибку в формате или логике данных.

Листинг 10: Функция validateBirthDate

```
bool MainWindow::validateBirthDate(const QString &date)
{
        QDate birthDate = QDate::fromString(date, "dd.
  MM.yyyy");
        QDate currentDate = QDate::currentDate();
        return (birthDate.isValid() &&
        birthDate < currentDate &&
        (birthDate.month() >= 1 && birthDate.month() <=
   12) &&
        (birthDate.day() >= 1 && birthDate.day() <=
  birthDate.daysInMonth()));
}
```

2.12 Функция validateEmail

Функция проверяет корректность формата электронной почты. Сначала она очищает строку, удаляя лишние пробелы вокруг символа "@"и перед ним, чтобы гарантировать, что адрес электронной почты введен правильно. Затем используется регулярное выражение для проверки, что адрес соответствует заданным критериям: он должен начинаться с буквы, содержать допустимые символы, содержать символ "@"и корректный домен, который должен заканчиваться на точку и недопустимый набор букв. Если адрес электронной почты соответствует этим условиям, функция возвращает ‘true‘, подтверждая правильность ввода, в противном случае — ‘false‘, что указывает на ошибку в формате.

Листинг 11: Функция validateEmail

```
bool MainWindow::validateEmail(const QString &email)
{
        QString cleanedEmail = email;
        cleanedEmail.replace(QRegularExpression("\\s*@(\\s*)"), "@");
        cleanedEmail.replace(QRegularExpression("(\\s*)@"), "@");
        cleanedEmail.replace(QRegularExpression("@(\\s*)"), "@");
        QRegularExpression regex("^[a-zA-Z][a-zA-Z0-9._%+-]*@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$");
        QRegularExpressionMatch match = regex.match(cleanedEmail);
        return match.hasMatch();
}
```

2.13 Функция removeSelectedRows

Функция удаляет выделенные строки из таблицы, управляя видимостью и очищая данные. Сначала она блокирует сигналы, чтобы предотвратить ненужные вызовы во время обработки. Затем проверяется, есть ли выделенные элементы; если нет, обработка заканчивается.

Далее создается множество для хранения строк, которые нужно удалить. Для каждого выделенного элемента проверяется, если он находится в последнем столбце (с зарплатой). Если это так, фон ячейки устанавливается в светло-голубой цвет, а если в другом столбце, текст ячейки просто очищается.

После этого функция проходит по всем строкам, чтобы очистить текст и фон для зарплат, и в конце обновляет столбец зарплат с помощью функ-

ции ‘updateSalaryColumn()‘, чтобы обеспечить актуальность данных. Функция завершает свою работу, разблокировав сигналы, возвращая таблицу к нормальному режиму работы.

Листинг 12: Функция removeSelectedRows

```
void MainWindow::removeSelectedRows()
{
        tableWidget->blockSignals(true);
        QList<QTableWidgetItem*> selectedItems = tableWidget->selectedItems();
        if (selectedItems.isEmpty()) {
                tableWidget->blockSignals(false);
                return;
        }
        QSet<int> rowsToDelete;
        for (QTableWidgetItem* item : selectedItems) {
                int row = item->row();
                if (item->column() == 9) {
                        item->setBackground(QColor(248, 248, 255));
                } else {
                        item->setText("");
                }
                rowsToDelete.insert(row);
        }
        for (int row : rowsToDelete) {
                QTableWidgetItem *salaryItem = tableWidget->item(row, 9);
                if (salaryItem) {
                        salaryItem->setText("");
                        salaryItem->setBackground(QColor(248, 248, 255));
                }
        }
        updateSalaryColumn();
        tableWidget->blockSignals(false);
}
```

2.14 Функция search

Функция осуществляет поиск по таблице на основе введенного текста, предоставляя пользователю возможность быстро находить записи. Сначала блокируются сигналы для предотвращения ненужных обновлений в процессе

обработки. Затем вводимый текст преобразуется в нижний регистр и очищается от пробелов.

Если текст введен, функция проходит по всем строкам и столбцам таблицы, проверяя, содержится ли текст поиска в каждой ячейке. Если содержимое ячейки соответствует запросу, эта строка отмечается желтым цветом для выделения, и строка остается видимой. В противном случае фон ячейки возвращается к светло-голубому, и строка скрывается.

В конечном итоге блокировка сигналов снимается, возвращая таблицу к нормальному режиму работы.

Листинг 13: Функция search

```
void MainWindow::search(const QString &searchText)
{
        tableWidget->blockSignals(true);

        QString loweredSearchText = searchText.toLower().trimmed();

        if (loweredSearchText.isEmpty()) {
                for (int row = 0; row < tableWidget->rowCount(); ++row) {
                        for (int col = 0; col < tableWidget->columnCount(); ++col) {
                                QTableWidgetItem *item = tableWidget->item(row, col);
                                if (item) {
                                        item->setBackground(QBrush(QColor(248, 248, 255)));

updateSalaryColumn();
                                }
                        }
                }
                for (int row = 0; row < tableWidget->rowCount(); ++row) {
                        tableWidget->setRowHidden(row, false);
                }
        } else {
                for (int row = 0; row < tableWidget->rowCount(); ++row) {
                        bool rowContainsSearchText = false;
```

```
23
24                          for (int col = 0; col < tableWidget->columnCount(); ++col) {
25                              QTableWidgetItem *item = tableWidget->item(row, col);
26                              if (item && item->text().toLower().contains(loweredSearchText)) {
27
                                rowContainsSearchText = true;
28                                  item->setBackground(QBrush(QColor(255, 255, 204)));
29                                  item->setForeground(QBrush(Qt::black));
30
                                updateSalaryColumn();
31                              } else {
32                                  item->setBackground(QBrush(QColor(248, 248, 255)));
33
                                updateSalaryColumn();
34                              }
35                          }
36                          tableWidget->setRowHidden(row, !rowContainsSearchText);
37                  }
38          }
39          tableWidget->blockSignals(false);
40  }
```

2.15 Функция cancelSearch

Функция отменяет поиск в таблице и возвращает все записи в их исходное состояние. Сначала она блокирует сигналы, чтобы избежать ненужных обновлений в процессе. Затем проходит по всем строкам и столбцам таблицы, сбрасывая фон каждой ячейки на светло-голубой цвет и устанавливая черный цвет текста, обеспечивая тем самым единообразный вид данных.

После этого функция очищает выделение, чтобы пользователь не видел, что какие-либо ячейки выбраны, возвращая интерфейс к нормальному состоянию. В конце разблокируются сигналы, позволяя таблице снова реагировать на изменения.

Листинг 14: Функция cancelSearch

```
1  void MainWindow::cancelSearch()
```

```
{
        tableWidget->blockSignals(true);
        for (int row = 0; row < tableWidget->rowCount()
  ; ++row) {
                for (int col = 0; col < tableWidget->
  columnCount(); ++col) {
                        QTableWidgetItem *item =
  tableWidget->item(row, col);
                        if (item) {
                                item->setBackground(
  QColor(248, 248, 255));
                                item->setForeground(
  QBrush(Qt::black));
                                updateSalaryColumn();
                        }
                }
        }
        tableWidget->clearSelection();
        tableWidget->blockSignals(false);
}
```

2.16 Функция loadDataFromFile

Функция загружает данные из выбранного CSV-файла в таблицу. Сначала открывается диалог для выбора файла, и если имя файла пустое, функция завершается. Затем открывается файл для чтения; если это не удается, отображается сообщение об ошибке.

Далее блокируются сигналы для предотвращения ненужных обновлений в процессе загрузки. Функция ищет первую пустую строку в таблице, чтобы начать вставку данных. Если пустых строк нет, создается новая строка в конце таблицы.

После этого функция читает файл построчно и разбивает каждую строку на отдельные поля по запятой. Каждое поле обрабатывается и добавляется в соответствующую ячейку таблицы. Каждая ячейка получает черный текст на светло-голубом фоне.

По завершении чтения файла функция закрывает файл, обновляет столбец зарплат, разблокирует сигналы, и информирует пользователя об успешной загрузке данных.

Листинг 15: Функция loadDataFromFile

```
void MainWindow::loadDataFromFile()
{
```

```
3       QString fileName = QFileDialog::getOpenFileName
    (this, "Загрузить данные", "", "CSV Files (*.csv);;All
    Files (*)");
4       if (fileName.isEmpty()) return;
5
6       QFile file(fileName);
7       if (!file.open(QIODevice::ReadOnly | QIODevice
    ::Text)) {
8               QMessageBox::warning(this, "Ошибка", "
    Не удалось открыть файл для чтения.");
9               return;
10      }
11      tableWidget->blockSignals(true);
12
13      int firstEmptyRow = -1;
14      for (int row = 0; row < tableWidget->rowCount()
    ; ++row) {
15              bool isEmpty = true;
16              for (int col = 0; col < tableWidget->
    columnCount(); ++col) {
17                      if (tableWidget->item(row, col)
     != nullptr && !tableWidget->item(row, col)->text().
    isEmpty()) {
18                              isEmpty = false;
19                              break;
20                      }
21              }
22              if (isEmpty) {
23                      firstEmptyRow = row;
24                      break;
25              }
26      }
27
28
29      if (firstEmptyRow == -1) {
30              firstEmptyRow = tableWidget->rowCount()
    ;
31              tableWidget->insertRow(firstEmptyRow);
32      }
33
34
35      while (!file.atEnd()) {
36              QByteArray line = file.readLine();
```

```
37                  QList<QByteArray> fields = line.split(',');
38
39                  for (int col = 0; col < fields.size(); ++col) {
40                      if (firstEmptyRow + col < tableWidget->rowCount()) {
41                          QTableWidgetItem *item = new QTableWidgetItem(QString(fields[col].trimmed()));
42                          item->setForeground(QBrush(Qt::black));
43                          item->setBackground(QBrush(QColor(248, 248, 255)));
44
45                          tableWidget->setItem(firstEmptyRow, col, item);
46                      }
47                  }
48                  firstEmptyRow++;
49          }
50
51          file.close();
52          updateSalaryColumn();
53          tableWidget->blockSignals(false);
54          QMessageBox::information(this, "Загрузка завершена", "Данные успешно загружены.");
55  }
```

2.17 Функция saveDataToFile

Функция сохраняет данные из таблицы в CSV-файл. Сначала открывается диалоговое окно для выбора имени файла для сохранения. Если имя файла пустое, функция завершает выполнение.

Затем создается объект ‘QFile‘, и происходит попытка открыть файл для записи. Если это не удается, пользователю показывается сообщение об ошибке.

Далее используется ‘QTextStream‘ для записи данных в файл. Функция проходит по всем строкам и столбцам таблицы, собирая данные каждой строки в ‘QStringList‘. Если ячейка содержит значение, оно добавляется в список.

После сбора данных строка формируется с помощью ‘rowData.join(’,’)‘, что объединяет элементы списка в строку, разделенную запятыми. Каждая

строка записывается в файл с переносом на новую строку.

По завершении записи файл закрывается, и пользователю отображается сообщение о успешном сохранении данных.

Листинг 16: Функция saveDataToFile

```
1         void MainWindow::saveDataToFile()
2         {
3                 QString fileName = QFileDialog::
  getSaveFileName(this, "Сохранить данные", "", "CSV 
  Files (*.csv);;All Files (*)");
4                 if (fileName.isEmpty()) return;
5                 QFile file(fileName);
6                 if (!file.open(QIODevice::WriteOnly |
  QIODevice::Text)) {
7                         QMessageBox::warning(this, "
  Ошибка", "Не удалось открыть файл для записи.");
8                         return;
9                 }
10                QTextStream out(&file);
11                for (int row = 0; row < tableWidget->
  rowCount(); ++row) {
12                        QStringList rowData;
13                        for (int col = 0; col <
  tableWidget->columnCount(); ++col) {
14                                QTableWidgetItem *item
  = tableWidget->item(row, col);
15                                if (item) {
16                                        rowData << item
  ->text();
17                                }
18                        }
19                        out << rowData.join(',') << "\n
  ";
20                }
21                file.close();
22                QMessageBox::information(this, "
  Сохранение завершено", "Данные успешно сохранены.");
23        }
```

2.18 Функция saveVisibleDataToFile

Функция ‘saveVisibleDataToFile()‘ сохраняет видимые данные из таблицы в CSV-файл, предоставляя пользователю диалог для выбора имени и места

сохранения. Если пользователь не указывает имя файла, функция завершает выполнение. После успешного открытия файла для записи, функция проходит по всем строкам таблицы, проверяя, какие из них видимы. Для каждой видимой строки собираются данные из ячеек в ‘QStringList‘, а затем записываются в файл, разделенные запятыми и заканчивающиеся переносом строки. В конце файл закрывается, и пользователю отображается сообщение об успешном сохранении данных.

Листинг 17: Функция saveVisibleDataToFile

```
1  void MainWindow::saveVisibleDataToFile()
2  {
3      QString fileName = QFileDialog::
   getSaveFileName(this, "Сохранить видимые данные при
   поиске", "", "CSV Files (*.csv);;All Files (*)");
4      if (fileName.isEmpty()) return;
5
6      QFile file(fileName);
7      if (!file.open(QIODevice::WriteOnly |
   QIODevice::Text)) {
8          QMessageBox::warning(this, "
   Ошибка", "Не удалось открыть файл для записи.");
9          return;
10     }
11
12     QTextStream out(&file);
13     for (int row = 0; row < tableWidget->
   rowCount(); ++row) {
14         if (!tableWidget->isRowHidden(
   row)) {
15             QStringList rowData;
16             for (int col = 0; col <
    tableWidget->columnCount(); ++col) {
17
   QTableWidgetItem *item = tableWidget->item(row, col)
   ;
18                 if (item) {
19                     rowData
    << item->text();
20                 }
21             }
22             out << rowData.join(',')
   ) << "\n";
23         }
```

```
                }
                file.close();
                QMessageBox::information(this, "Сохранение завершено", "Видимые данные успешно сохранены.");
        }
```

2.19 Функция sortTableByColumn

Функция ‘sortTableByColumn(int column)‘ предназначена для сортировки таблицы по выбранному столбцу. Сначала она блокирует сигналы, чтобы предотвратить ненужные обновления в процессе сортировки. Затем устанавливается порядок сортировки — в данном случае, по убыванию. После этого вызывается метод ‘sortItems()‘, который выполняет сортировку данных в указанном столбце. Также устанавливается указатель сортировки на заголовке столбца, чтобы пользователь видел, по какому столбцу идет сортировка. Наконец, блокировка сигналов снимается, возвращая таблицу к нормальному режиму работы. Эта функция обеспечивает быструю и удобную сортировку записей в таблице.

Листинг 18: Функция sortTableByColumn

```
void MainWindow::sortTableByColumn(int column)
{
        tableWidget->blockSignals(true);
        Qt::SortOrder order = Qt::DescendingOrder;
        tableWidget->sortItems(column, order);
        tableWidget->horizontalHeader()->setSortIndicator(column, order);
        tableWidget->blockSignals(false);
}
```

3 Тестирование приложения

После запуска пользователь видит основное окно программы – рис. 2.

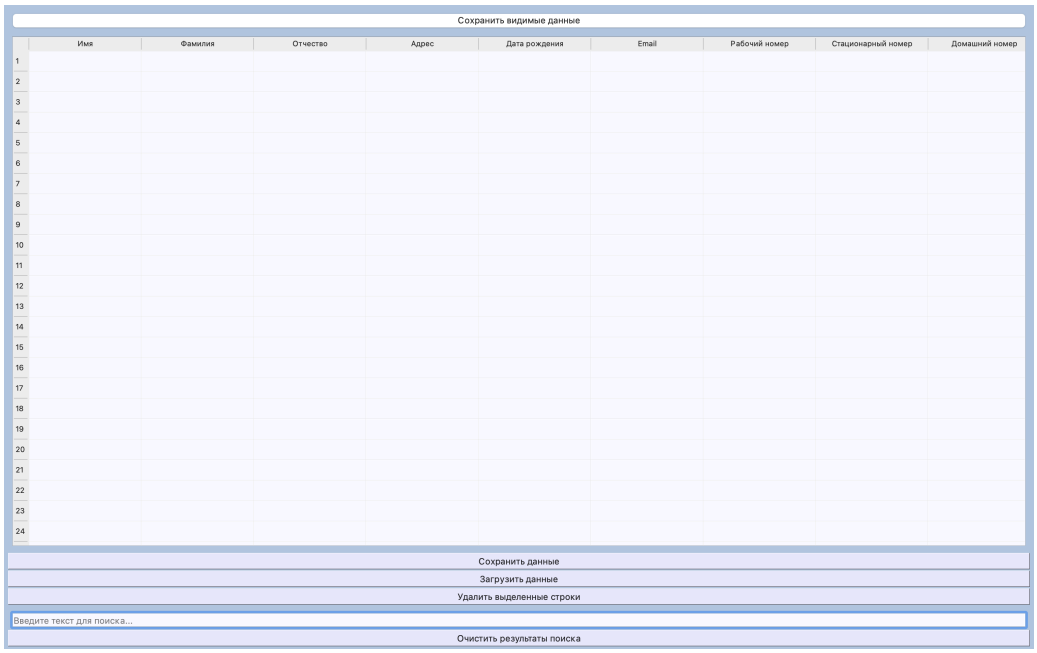

Рис. 2: Начальное меню

При нажатии кнопки «Загрузить данные» пользователь выбирает файл с данными для загрузки – рис. 3.

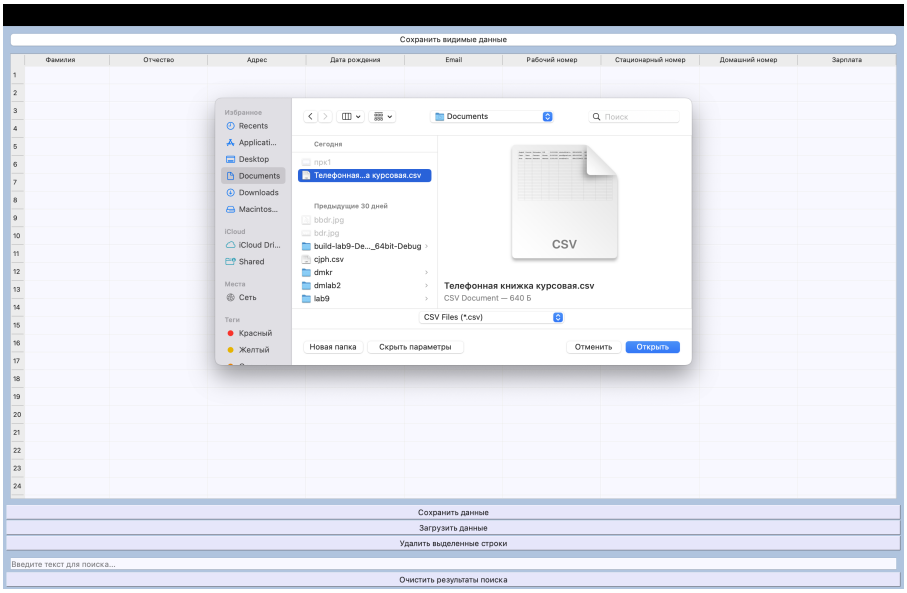

Рис. 3: Загрузка данных

При успешной загрузке данных высвечивается уведомление – рис. 4.

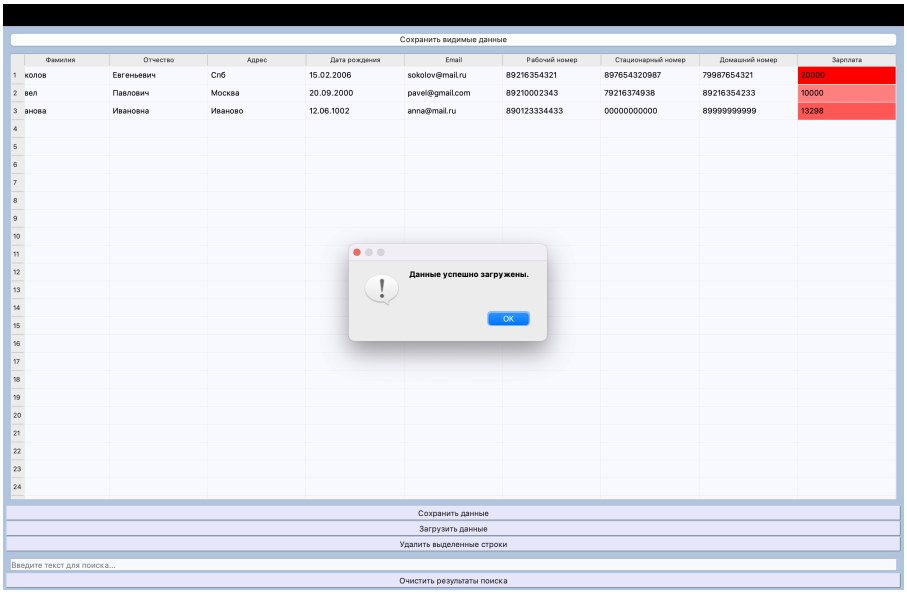

Рис. 4: Успешная загрузка данных

При желании пользователь может сохранить данные в файл нажав «Сохранить данные» – рис. 5.

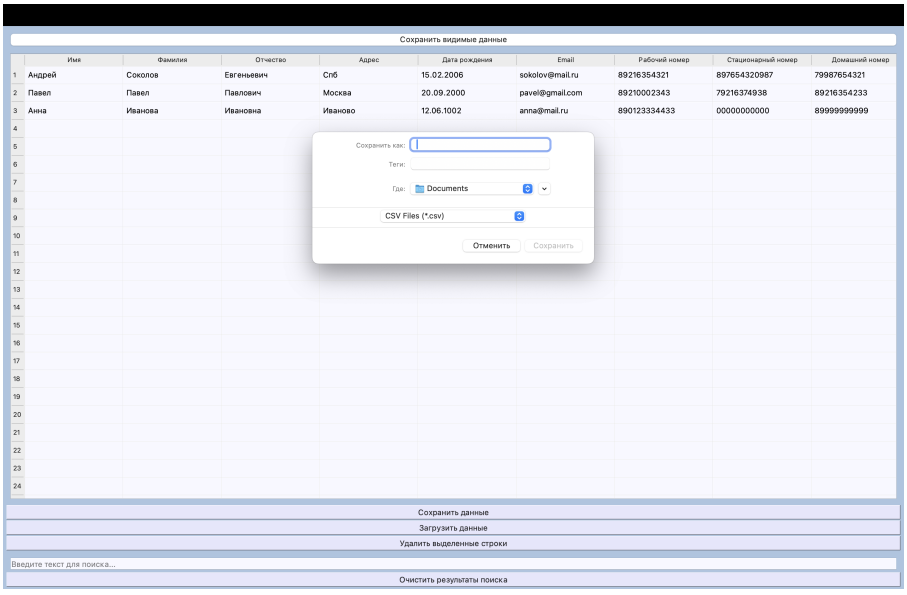

Рис. 5: Сохранение данных

При поиске остаются только строки с подходящими данными, ячейки подсвечиваются желтым – рис. 6.

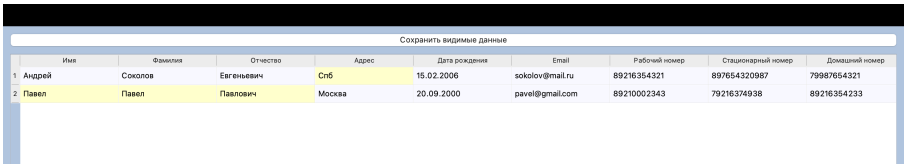

Рис. 6: Поиск

Можно сохранить только видимые при поиске строки нажав кнопку «Сохранить видимые данные» – рис. 7.

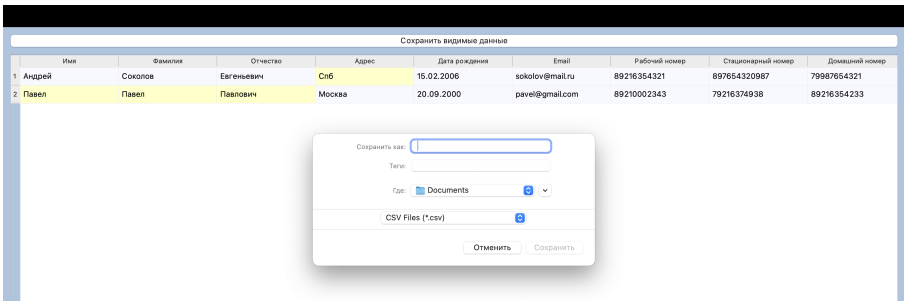

Рис. 7: Сохранить видимые данные

Для удаления данных нужно выделить нужные ячейки и нажать кнопку «Удалить выделенные строки» – рис. 8, рис. 9.

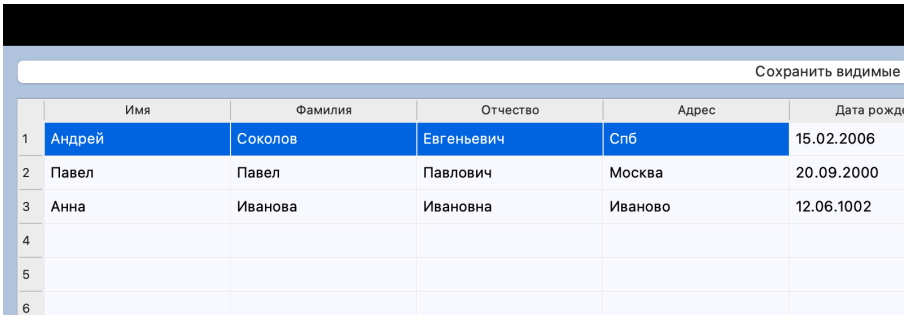

Рис. 8: Удалить выделенные строки

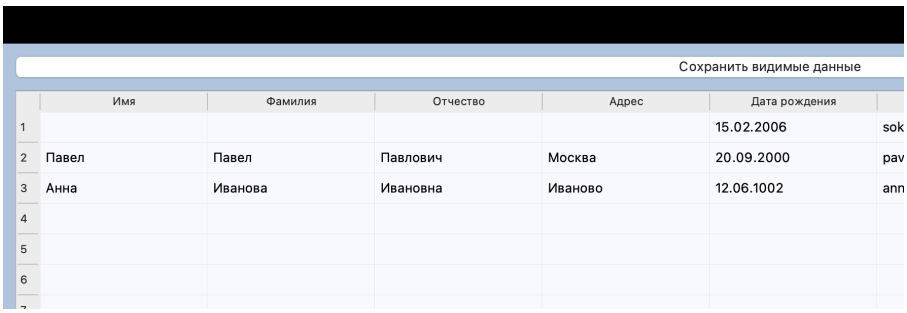

Рис. 9: Удалить выделенные строки

При изменении зарплаты ячейки меняют свой цвет в соответствии – рис. 10 рис. 11.

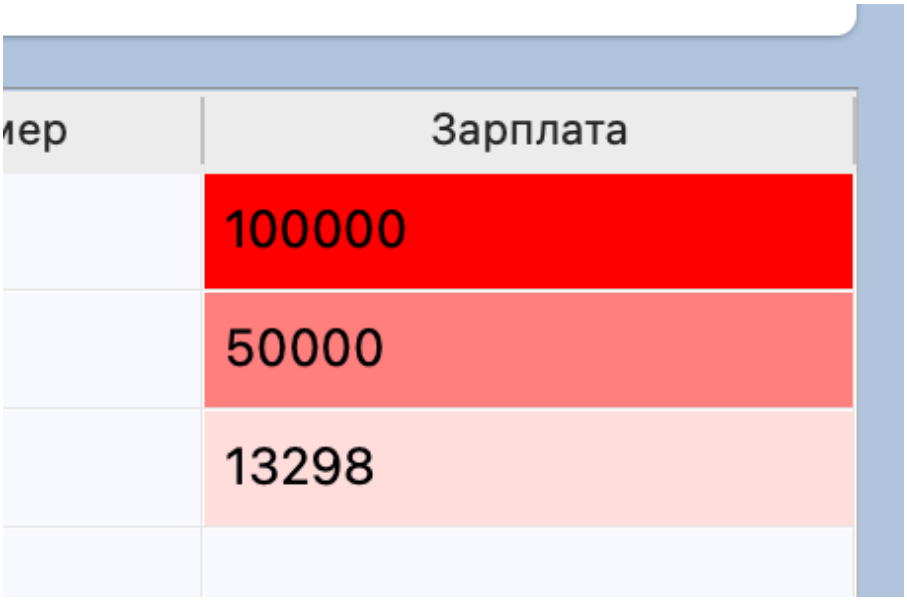

Рис. 10: Изменение зарплаты

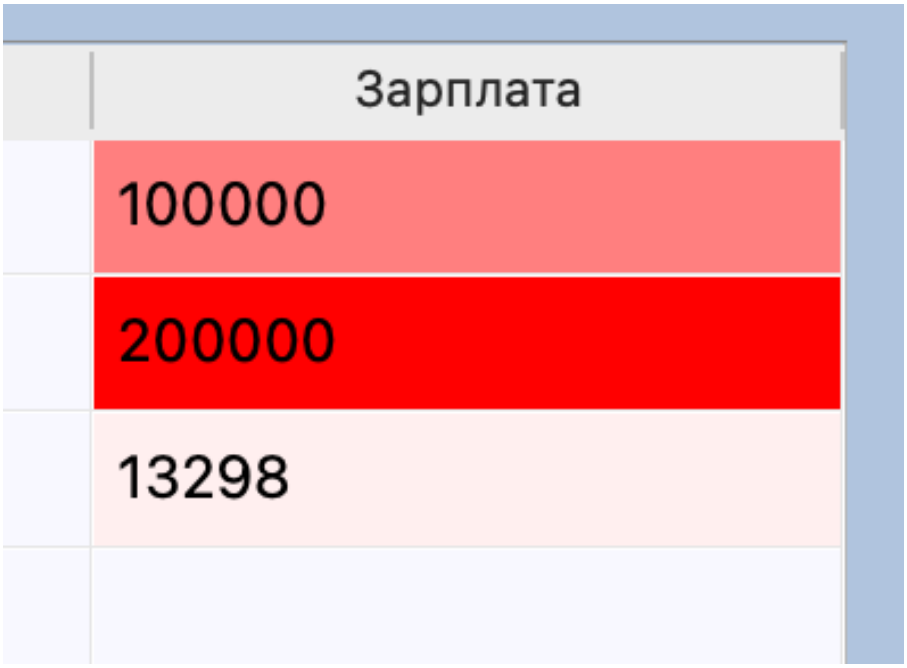

Рис. 11: Изменение зарплаты

Пользователь может отсортировать данные по выбранному столбцу по убыванию – рис. 12, рис. 13.

	Имя	Фамилия	Отчество	Адрес	Дата рождения	Email	Рабочий номер	Стационарный номер	Домашний номер
1	Андрей	Соколов	Евгеньевич	Спб	15.02.2006	sokolov@mail.ru	89216354321	897654320987	79987654321
2	Павел	Павел	Павлович	Москва	20.09.2000	pavel@gmail.com	89210002343	79216374938	89216354233
3	Анна	Иванова	Ивановна	Иваново	12.06.1002	anna@mail.ru	890123334433	00000000000	89999999999
4									

Рис. 12: Неотсортированная телефонная книжка

	Имя	Фамилия	Отчество	Адрес	Дата рождения	Email	Рабочий номер	Стационарный номер	Домашний номер
1	Андрей	Соколов	Евгеньевич	Спб	15.02.2006	sokolov@mail.ru	89216354321	897654320987	79987654321
2	Павел	Павел	Павлович	Москва	20.09.2000	pavel@gmail.com	89210002343	79216374938	89216354233
3	Анна	Иванова	Ивановна	Иваново	12.06.1002	anna@mail.ru	890123334433	00000000000	89999999999

Рис. 13: Отсортированная по фамилии телефонная книжка

Несколько примеров предупреждения о некорректном вводе – рис 14, рис 15, рис 16.

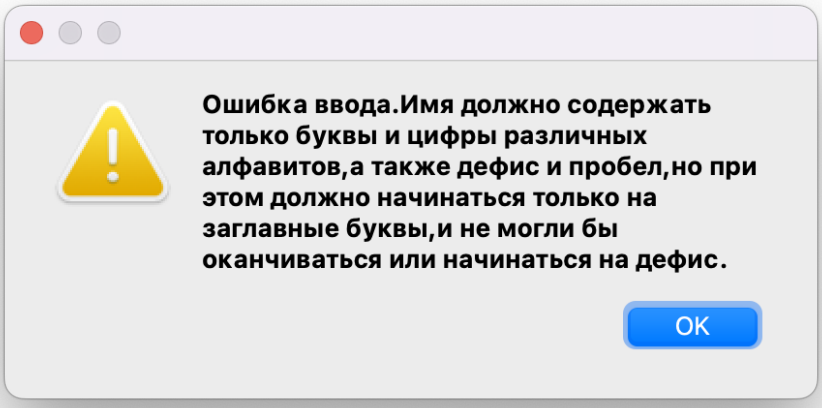

Рис. 14: Некорректный ввод имени

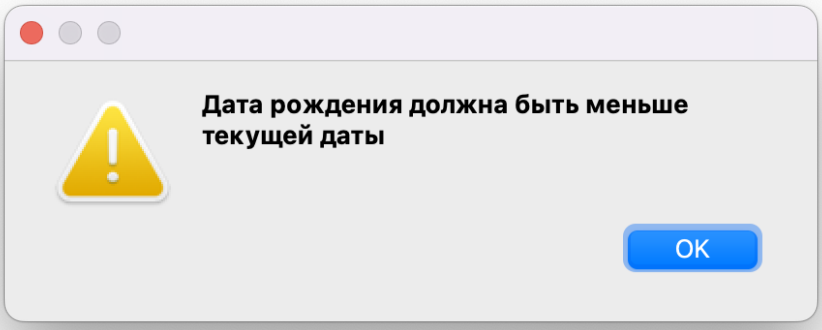

Рис. 15: Некорректный ввод даты рождения

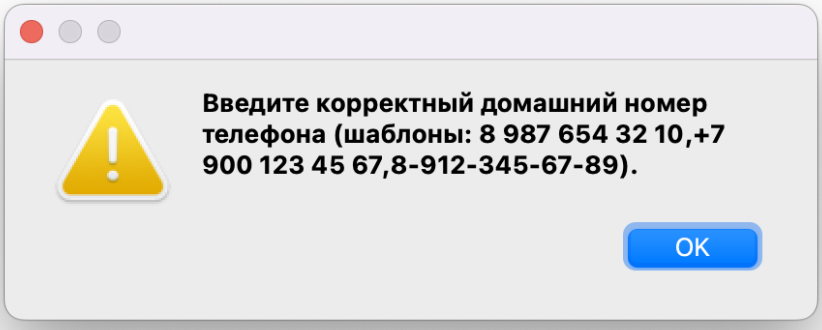

Рис. 16: Некорректный ввод домашнего номера

Заключение

В результате работы все поставленные задачи были выполнены, а также был добавлен дополнительный функционал в приложение (столбик с зарплатой). Реализованное приложение работает корректно и правильно выполняет все функции.

Код приложения состоит из 500 строчек. Работа выполнена в Qt Creator.

В качестве дополнительного функционала можно добавить возможность создавать свои столбцы пользователю по отдельной кнопке.

Таким образом, успешно выполненные задачи не только привели к созданию функционального приложения, но и стали ценным опытом в освоении технологий программирования и разработки пользовательских интерфейсов. Проект позволил развить навыки работы с Qt и углубить понимание принципов создания кроссплатформенных приложений.

Приложение А. Исходный код

А1. Исходный код файла mainwindow.h

```
#pragma once
#include <QMainWindow>
#include <QTableWidget>
#include <QRegularExpressionValidator>
#include <QMessageBox>
#include <QRegularExpression>
#include <QPushButton>
#include <QDate>

class MainWindow : public QMainWindow
{
        Q_OBJECT

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

        private:
        void search(const QString &searchText);
        void loadDataFromFile();
        void saveDataToFile();
        void cancelSearch();
        void closeWin();
        void removeSelectedRows();
        void setupTable();
        void setupValidators();
        void handleCellChanged(int row, int column);
        void saveVisibleDataToFile();
        void updateSalaryColumn();
        void sortTableByColumn(int column);

        bool validateName(const QString &name);
        bool validateBirthDate(const QString &date);
        bool validateEmail(const QString &email);
        bool validatePhoneNumber(QString &phoneNumber);
        bool validateHome(QString &homePhoneNumber);
        bool validateStat(QString &statPhoneNumber);

        QString formattedPhone;
```

```
40          QLineEdit *searchLineEdit;
41          QTableWidget *tableWidget;
42          QRegularExpressionValidator *phoneValidator;
43          QRegularExpressionValidator *homePhoneValidator
    ;
44          QRegularExpressionValidator *statPhoneValidator
    ;
45  };
```

А2. Исходный код файла mainwindow.cpp

```
#include "mainwindow.h"
#include <QVBoxLayout>
#include <QWidget>
#include <QMessageBox>
#include <QRegularExpression>
#include <QDate>
#include <QSet>
#include <QLineEdit>
#include <QFileDialog>
#include <QTextStream>
#include <QHeaderView>

MainWindow::MainWindow(QWidget *parent)
: QMainWindow(parent), tableWidget(new QTableWidget(
  this)), searchLineEdit(new QLineEdit(this))
{
        setupTable();
        setupValidators();
        searchLineEdit->setPlaceholderText("Введите␣
  текст␣для␣поиска...");
        connect(searchLineEdit, &QLineEdit::textChanged
  , this, [this]() {
                search(searchLineEdit->text());
        });
        QPushButton *deleteButton = new QPushButton("
  Удалить␣выделенные␣строки", this);
        deleteButton->setStyleSheet("background-color:␣
  #E6E6FA");
        connect(deleteButton, &QPushButton::clicked,
  this, &MainWindow::removeSelectedRows);

        QPushButton *loadButton = new QPushButton("
  Загрузить␣данные", this);
        loadButton->setStyleSheet("background-color:␣#
  E6E6FA");
        connect(loadButton, &QPushButton::clicked, this
  , &MainWindow::loadDataFromFile);

        QPushButton *saveButton = new QPushButton("
  Сохранить␣данные", this);
```

```
31          saveButton->setStyleSheet("background-color: #
    E6E6FA");
32          connect(saveButton, &QPushButton::clicked, this
    , &MainWindow::saveDataToFile);
33
34          QWidget *centralWidget = new QWidget(this);
35          QVBoxLayout *layout = new QVBoxLayout(
    centralWidget);
36          QPushButton *cancelSearchButton = new
    QPushButton("Очистить результаты поиска", this);
37          cancelSearchButton->setStyleSheet("background-
    color: #E6E6FA");
38          connect(cancelSearchButton, &QPushButton::
    clicked, this, &MainWindow::cancelSearch);
39
40          QPushButton *saveVisibleButton = new
    QPushButton("Сохранить видимые данные", this);
41          connect(saveVisibleButton, &QPushButton::
    clicked, this, &MainWindow::saveVisibleDataToFile);
42          layout->addWidget(saveVisibleButton);
43
44
45          layout->addWidget(tableWidget);
46          layout->addWidget(saveButton);
47          layout->addWidget(loadButton);
48          layout->addWidget(deleteButton);
49          setCentralWidget(centralWidget);
50          layout->addWidget(searchLineEdit);
51          searchLineEdit->setStyleSheet("background-color
    : #F8F8FF");
52          layout->addWidget(cancelSearchButton);
53          connect(tableWidget, &QTableWidget::cellChanged
    , this, &MainWindow::handleCellChanged);
54
55          QPalette Pal(palette());
56          Pal.setColor(QPalette::Background, QColor(176,
    196, 222));
57          this->setAutoFillBackground(true);
58          this->setPalette(Pal);
59          connect(searchLineEdit, &QLineEdit::textChanged
    , this, [this]() {
60                  search(searchLineEdit->text());
61          });
```

```
62
63          tableWidget->setSortingEnabled(false);
64
65          // Добавьте обработчик для заголовков таблицы
66          connect(tableWidget->horizontalHeader(), &
    QHeaderView::sectionClicked, this, &MainWindow::
    sortTableByColumn);
67
68  }
69
70  MainWindow::~MainWindow()
71  {
72          delete phoneValidator;
73          delete homePhoneValidator;
74          delete statPhoneValidator;
75  }
76
77  void MainWindow::setupTable()
78  {
79          tableWidget->setRowCount(30);
80          tableWidget->setColumnCount(10);
81          QStringList headers {"Имя", "Фамилия", "Отчество",
     "Адрес", "Дата␣рождения", "Email", "Рабочий␣номер", "
    Стационарный␣номер", "Домашний␣номер", "Зарплата"};
82          tableWidget->setHorizontalHeaderLabels(headers)
    ;
83          tableWidget->setSortingEnabled(true);
84          for (int col = 0; col < tableWidget->
    columnCount(); ++col) {
85                  tableWidget->setColumnWidth(col, 165);
86          }
87          for (int row = 0; row < tableWidget->rowCount()
    ; ++row) {
88                  for (int col = 0; col < tableWidget->
    columnCount(); ++col) {
89                          QTableWidgetItem *item = new
    QTableWidgetItem();
90                          item->setBackground(QColor(248,
     248, 255));
91                          item->setForeground(QBrush(Qt::
    black));
92                          tableWidget->setItem(row, col,
    item);
```

```
93                  }
94          }
95  }
96
97  void MainWindow::updateSalaryColumn()
98  {
99          int maxSalary = 0;
100         for (int row = 0; row < tableWidget->rowCount()
    ; ++row) {
101                 QTableWidgetItem *salaryItem =
    tableWidget->item(row, 9);
102                 if (salaryItem) {
103                         bool ok;
104                         int salary = salaryItem->text()
    .toInt(&ok);
105                         if (ok && salary > maxSalary) {
106                                 maxSalary = salary;
107                         }
108                 }
109         }
110         for (int row = 0; row < tableWidget->rowCount()
    ; ++row) {
111                 QTableWidgetItem *salaryItem =
    tableWidget->item(row, 9);
112                 if (salaryItem) {
113                         bool ok;
114                         int salary = salaryItem->text()
    .toInt(&ok);
115                         if (ok && maxSalary > 0) {
116                                 int percentage = (
    static_cast<float>(salary) / maxSalary) * 100;
117                                 QColor fillColor =
    QColor(255, 255 * (100 - percentage) / 100, 255 *
    (100 - percentage) / 100);
118                                 salaryItem->
    setBackground(fillColor);
119                         }
120                 }
121         }
122 }
123
124 void MainWindow::setupValidators()
125 {
```

```
126         phoneValidator = new
    QRegularExpressionValidator(QRegularExpression("
    ^(8|\\+7)\\s?\\$?\\d{3}\\$?\\s?\\d{3}[\\s-]?\\d
    {2}[\\s-]?\\d{2}$"), this);
127         homePhoneValidator = new
    QRegularExpressionValidator(QRegularExpression("
    ^(8|\\+7)\\s?\\$?\\d{3}\\$?\\s?\\d{3}[\\s-]?\\d
    {2}[\\s-]?\\d{2}$"), this);
128         statPhoneValidator = new
    QRegularExpressionValidator(QRegularExpression("
    ^(8|\\+7)\\s?\\$?\\d{3}\\$?\\s?\\d{3}[\\s-]?\\d
    {2}[\\s-]?\\d{2}$"), this);
129 }
130
131 void MainWindow::handleCellChanged(int row, int column)
132 {
133         tableWidget->blockSignals(true);
134         if (!tableWidget->item(row, column)) {
135                 return;
136         }
137         QString text = tableWidget->item(row, column)->
    text().trimmed();
138         text.replace(QRegularExpression("\\s+"), "␣");
139         switch (column) {
140                 case 0:
141                 if (!validateName(text)) {
142                         QMessageBox::warning(this, "
    Ошибка", "Ошибка␣ввода.␣Имя␣должно␣содержать␣только␣буквы␣и␣
    цифры␣различных␣алфавитов,␣а␣также␣дефис␣и␣пробел,␣но␣при␣
    этом␣должно␣начинаться␣только␣на␣заглавные␣буквы,␣и␣не␣могли␣
    бы␣оканчиваться␣или␣начинаться␣на␣дефис.");
143                         tableWidget->item(row, column)
    ->setText("");
144                 } else {
145                         tableWidget->item(row, column)
    ->setText(text);
146                 }
147                 break;
148
149                 case 1:
150                 if (!validateName(text)) {
151                         QMessageBox::warning(this, "
    Ошибка", "Ошибка␣ввода.␣Фамилия␣должна␣содержать␣только␣
```

```
буквы␣и␣цифры␣различных␣алфавитов,␣а␣также␣дефис␣и␣пробел,␣
но␣при␣этом␣должно␣начинаться␣только␣на␣заглавные␣буквы,␣и␣
не␣могли␣бы␣оканчиваться␣или␣начинаться␣на␣дефис.");
152                         tableWidget->item(row, column)
->setText("");
153                 } else {
154                         tableWidget->item(row, column)
->setText(text);
155                 }
156                 break;
157
158                 case 2:
159                 if (!validateName(text)) {
160                         QMessageBox::warning(this, "
Ошибка", "Ошибка␣ввода.␣Отчество␣должно␣содержать␣только␣
буквы␣и␣цифры␣различных␣алфавитов,␣а␣также␣дефис␣и␣пробел,␣
но␣при␣этом␣должно␣начинаться␣только␣на␣заглавные␣буквы,␣и␣
не␣могли␣бы␣оканчиваться␣или␣начинаться␣на␣дефис.");
161                         tableWidget->item(row, column)
->setText("");
162                 } else {
163                         tableWidget->item(row, column)
->setText(text);
164                 }
165                 break;
166
167                 case 4:
168                 if (!validateBirthDate(text)) {
169                         QMessageBox::warning(this, "
Ошибка", "Дата␣рождения␣должна␣быть␣меньше␣текущей␣даты");
170                         tableWidget->item(row, column)
->setText("");
171                 } else {
172                         tableWidget->item(row, column)
->setText(text);
173                 }
174                 break;
175
176                 case 5:
177                 if (!validateEmail(text)) {
178                         QMessageBox::warning(this, "
Ошибка", "E-mail␣должен␣содержать␣в␣себе␣имя␣пользователя␣
состоящее␣из␣латинских␣букв␣и␣цифр,␣символ␣разделения␣
```

```
пользователя и имени домена(@), а также сам домен состоящий
из латинских букв и цифр. Все незначащие пробелы должны быть
удалены.");
179                         tableWidget->item(row, column)
->setText("");
180                 } else {
181                         tableWidget->item(row, column)
->setText(text);
182                 }
183                 break;
184
185                 case 6:
186                 if (!validatePhoneNumber(text)) {
187                         tableWidget->item(row, column)
->setText("");
188                 } else {
189                         tableWidget->item(row, column)
->setText(text);
190                 }
191                 break;
192                 case 7:
193                 if (!validateHome(text)) {
194                         tableWidget->item(row, column)
->setText("");
195                 } else {
196                         tableWidget->item(row, column)
->setText(text);
197                 }
198                 break;
199                 case 8:
200                 if (!validateStat(text)) {
201                         tableWidget->item(row, column)
->setText("");
202                 } else {
203                         tableWidget->item(row, column)
->setText(text);
204                 }
205                 break;
206                 case 9:
207                 if(column==9){
208                         updateSalaryColumn();
209                 }
210                 default:
```

```
211                     break;
212             }
213             tableWidget->blockSignals(false);
214     }
215
216     bool MainWindow::validateHome(QString &homephoneNumber)
217     {
218             QString cleanedPhone = homephoneNumber;
219             cleanedPhone.remove(QRegularExpression("[^0-9]"
        ));
220             if (cleanedPhone.length() < 10) {
221                     QMessageBox::warning(this, "Ошибка␣
        ввода", "Введите␣корректный␣домашний␣номер␣телефона␣шаблоны(:
        ␣8␣987␣654␣32␣10,␣+7␣900␣123␣45␣67,␣8-912-345-67-89)
        .");
222                     return false;
223             }
224             homephoneNumber = cleanedPhone;
225             return true;
226     }
227
228     bool MainWindow::validateStat(QString &statphoneNumber)
229     {
230             QString cleanedPhone = statphoneNumber;
231             cleanedPhone.remove(QRegularExpression("[^0-9]"
        ));
232             if (cleanedPhone.length() < 10) {
233                     QMessageBox::warning(this, "Ошибка␣
        ввода", "Введите␣корректный␣стационарный␣номер␣телефона␣
        шаблоны(:␣8␣987␣654␣32␣10,␣+7␣900␣123␣45␣67,␣
        8-912-345-67-89).");
234                     return false;
235             }
236             statphoneNumber = cleanedPhone;
237             return true;
238     }
239
240     bool MainWindow::validateName(const QString &name)
241     {
242             QRegularExpression regex("^[A-AZЯЁ-][A-Za-Az
        Яаяё--0-9]*(?:[␣-][A-Za-AzЯаяё--0-9]+)*[A-Za-Az
        Яаяё--0-9]$");
243             return regex.match(name).hasMatch();
```

```
244 }
245
246 bool MainWindow::validateBirthDate(const QString &date)
247 {
248         QDate birthDate = QDate::fromString(date, "dd.
      MM.yyyy");
249         QDate currentDate = QDate::currentDate();
250         return (birthDate.isValid() &&
251         birthDate < currentDate &&
252         (birthDate.month() >= 1 && birthDate.month() <=
       12) &&
253         (birthDate.day() >= 1 && birthDate.day() <=
      birthDate.daysInMonth()));
254 }
255
256 bool MainWindow::validateEmail(const QString &email)
257 {
258         QString cleanedEmail = email;
259         cleanedEmail.replace(QRegularExpression("\\s*@
      (\\s*)"), "@");
260         cleanedEmail.replace(QRegularExpression("(\\s*)
      @"), "@");
261         cleanedEmail.replace(QRegularExpression("@(\\s
      *)"), "@");
262         QRegularExpression regex("^[a-zA-Z][a-zA-Z0-9._
      %+-]*@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$");
263         QRegularExpressionMatch match = regex.match(
      cleanedEmail);
264         return match.hasMatch();
265 }
266
267 bool MainWindow::validatePhoneNumber(QString &
      phoneNumber)
268 {
269         QString cleanedPhone = phoneNumber;
270         cleanedPhone.remove(QRegularExpression("[^0-9]"
      ));
271         if (cleanedPhone.length() < 10) {
272                 QMessageBox::warning(this, "Ошибка␣
      ввода", "Введите␣корректный␣рабочий␣номер␣телефона␣шаблоны(:␣
      8␣987␣654␣32␣10,␣+7␣900␣123␣45␣67,␣8-912-345-67-89).
      ");
273                 return false;
```

```
274             }
275             phoneNumber = cleanedPhone;
276             return true;
277     }
278
279     void MainWindow::removeSelectedRows()
280     {
281             tableWidget->blockSignals(true);
282             QList<QTableWidgetItem*> selectedItems =
        tableWidget->selectedItems();
283             if (selectedItems.isEmpty()) {
284                     tableWidget->blockSignals(false);
285                     return;
286             }
287             QSet<int> rowsToDelete;
288             for (QTableWidgetItem* item : selectedItems) {
289                     int row = item->row();
290                     if (item->column() == 9) {
291                             item->setBackground(QColor(248,
         248, 255));
292                     } else {
293                             item->setText("");
294                     }
295                     rowsToDelete.insert(row);
296             }
297             for (int row : rowsToDelete) {
298                     QTableWidgetItem *salaryItem =
        tableWidget->item(row, 9);
299                     if (salaryItem) {
300                             salaryItem->setText("");
301                             salaryItem->setBackground(
        QColor(248, 248, 255));
302                     }
303             }
304             updateSalaryColumn();
305             tableWidget->blockSignals(false);
306     }
307
308
309     void MainWindow::search(const QString &searchText)
310     {
311             tableWidget->blockSignals(true);
312
```

```
313         QString loweredSearchText = searchText.toLower
    ().trimmed();
314 
315         if (loweredSearchText.isEmpty()) {
316                 for (int row = 0; row < tableWidget->
    rowCount(); ++row) {
317                         for (int col = 0; col <
    tableWidget->columnCount(); ++col) {
318                                 QTableWidgetItem *item
    = tableWidget->item(row, col);
319                                 if (item) {
320                                         item->
    setBackground(QBrush(QColor(248, 248, 255)));
321 
    updateSalaryColumn();
322                                 }
323                         }
324                 }
325                 for (int row = 0; row < tableWidget->
    rowCount(); ++row) {
326                         tableWidget->setRowHidden(row,
    false);
327                 }
328         } else {
329                 for (int row = 0; row < tableWidget->
    rowCount(); ++row) {
330                         bool rowContainsSearchText =
    false;
331 
332                         for (int col = 0; col <
    tableWidget->columnCount(); ++col) {
333                                 QTableWidgetItem *item
    = tableWidget->item(row, col);
334                                 if (item && item->text
    ().toLower().contains(loweredSearchText)) {
335 
    rowContainsSearchText = true;
336                                         item->
    setBackground(QBrush(QColor(255, 255, 204)));
337                                         item->
    setForeground(QBrush(Qt::black));
338 
    updateSalaryColumn();
```

```
339                                     } else {
340                                             item->
    setBackground(QBrush(QColor(248, 248, 255)));
341
    updateSalaryColumn();
342                                     }
343                             }
344                             tableWidget->setRowHidden(row,
    !rowContainsSearchText);
345                     }
346             }
347             tableWidget->blockSignals(false);
348 }
349
350
351
352 void MainWindow::cancelSearch()
353 {
354             tableWidget->blockSignals(true);
355             for (int row = 0; row < tableWidget->rowCount()
    ; ++row) {
356                     for (int col = 0; col < tableWidget->
    columnCount(); ++col) {
357                             QTableWidgetItem *item =
    tableWidget->item(row, col);
358                             if (item) {
359                                     item->setBackground(
    QColor(248, 248, 255));
360                                     item->setForeground(
    QBrush(Qt::black));
361                                     updateSalaryColumn();
362                             }
363                     }
364             }
365             tableWidget->clearSelection();
366             tableWidget->blockSignals(false);
367 }
368
369 void MainWindow::loadDataFromFile()
370 {
371             QString fileName = QFileDialog::getOpenFileName
    (this, "Загрузить данные", "", "CSV Files (*.csv);;All
    Files (*)");
```

```
372         if (fileName.isEmpty()) return;
373 
374         QFile file(fileName);
375         if (!file.open(QIODevice::ReadOnly | QIODevice
    ::Text)) {
376                 QMessageBox::warning(this, "Ошибка", "
    Не удалось открыть файл для чтения.");
377                 return;
378         }
379         tableWidget->blockSignals(true);
380 
381         int firstEmptyRow = -1;
382         for (int row = 0; row < tableWidget->rowCount()
    ; ++row) {
383                 bool isEmpty = true;
384                 for (int col = 0; col < tableWidget->
    columnCount(); ++col) {
385                         if (tableWidget->item(row, col)
     != nullptr && !tableWidget->item(row, col)->text().
    isEmpty()) {
386                                 isEmpty = false;
387                                 break;
388                         }
389                 }
390                 if (isEmpty) {
391                         firstEmptyRow = row;
392                         break;
393                 }
394         }
395 
396 
397         if (firstEmptyRow == -1) {
398                 firstEmptyRow = tableWidget->rowCount()
    ;
399                 tableWidget->insertRow(firstEmptyRow);
400         }
401 
402 
403         while (!file.atEnd()) {
404                 QByteArray line = file.readLine();
405                 QList<QByteArray> fields = line.split('
    ,');
406 
```

```
407                 for (int col = 0; col < fields.size();
    ++col) {
408                     if (firstEmptyRow + col <
    tableWidget->rowCount()) {
409                         QTableWidgetItem *item
    = new QTableWidgetItem(QString(fields[col].trimmed()
    ));
410                         item->setForeground(
    QBrush(Qt::black));
411                         item->setBackground(
    QBrush(QColor(248, 248, 255)));
412
413                         tableWidget->setItem(
    firstEmptyRow, col, item);
414                     }
415                 }
416                 firstEmptyRow++;
417         }
418
419         file.close();
420         updateSalaryColumn();
421         tableWidget->blockSignals(false);
422         QMessageBox::information(this, "Загрузка␣
    завершена", "Данные␣успешно␣загружены.");
423 }
424
425
426
427 void MainWindow::saveDataToFile()
428 {
429         QString fileName = QFileDialog::getSaveFileName
    (this, "Сохранить␣данные", "", "CSV␣Files␣(*.csv);;All␣
    Files␣(*)");
430         if (fileName.isEmpty()) return;
431         QFile file(fileName);
432         if (!file.open(QIODevice::WriteOnly | QIODevice
    ::Text)) {
433                 QMessageBox::warning(this, "Ошибка", "
    Не␣удалось␣открыть␣файл␣для␣записи.");
434                 return;
435         }
436         QTextStream out(&file);
```

```
437         for (int row = 0; row < tableWidget->rowCount()
    ; ++row) {
438                 QStringList rowData;
439                 for (int col = 0; col < tableWidget->
    columnCount(); ++col) {
440                         QTableWidgetItem *item =
    tableWidget->item(row, col);
441                         if (item) {
442                                 rowData << item->text()
    ;
443                         }
444                 }
445                 out << rowData.join(',') << "\n";
446         }
447         file.close();
448         QMessageBox::information(this, "Сохранение␣
    завершено", "Данные␣успешно␣сохранены.");
449 }
450
451 void MainWindow::saveVisibleDataToFile()
452 {
453         QString fileName = QFileDialog::getSaveFileName
    (this, "Сохранить␣видимые␣данные␣при␣поиске", "", "CSV␣
    Files␣(*.csv);;All␣Files␣(*)");
454         if (fileName.isEmpty()) return;
455
456         QFile file(fileName);
457         if (!file.open(QIODevice::WriteOnly | QIODevice
    ::Text)) {
458                 QMessageBox::warning(this, "Ошибка", "
    Не␣удалось␣открыть␣файл␣для␣записи.");
459                 return;
460         }
461
462         QTextStream out(&file);
463         for (int row = 0; row < tableWidget->rowCount()
    ; ++row) {
464                 if (!tableWidget->isRowHidden(row)) {
465                         QStringList rowData;
466                         for (int col = 0; col <
    tableWidget->columnCount(); ++col) {
467                                 QTableWidgetItem *item
    = tableWidget->item(row, col);
```

```
468                                     if (item) {
469                                         rowData << item
    ->text();
470                                     }
471                                 }
472                                 out << rowData.join(',') << "\n
    ";
473                             }
474                         }
475                         file.close();
476                         QMessageBox::information(this, "Сохранение
    завершено", "Видимые данные успешно сохранены.");
477 }
478
479 void MainWindow::sortTableByColumn(int column)
480 {
481                         tableWidget->blockSignals(true);
482                         Qt::SortOrder order = Qt::DescendingOrder;
483                         tableWidget->sortItems(column, order);
484                         tableWidget->horizontalHeader()->
    setSortIndicator(column, order);
485                         tableWidget->blockSignals(false);
486 }
```

А3. Исходный код файла main.cpp

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

int main(int argc, char *argv[]) {
        QApplication app(argc, argv);

        MainWindow window;
        window.resize(800, 500);
        window.show();

        return app.exec();
}

```