МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»
(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова)

Кафедра О7 Информационные системы и программная инженерия
шифр наименование кафедры, по которой выполняется работа

Дисциплина Визуальное программирование
наименование дисциплины

ПРАКТИЧЕСКАЯ РАБОТА № 3

Работа с файлами XML/JSON

Язык: C++ Qt

Вариант №10

ОБУЧАЮЩИЙСЯ

группы О726Б

Махов Н.М.
подпись фамилия и инициалы

дата сдачи

ПРОВЕРИЛ

Преподаватель
ученая степень, ученое звание, должность

Устиновский Г.С.
подпись фамилия и инициалы

Оценка / балльная оценка

дата проверки

г. Санкт-Петербург
2024 г.

СОДЕРЖАНИЕ

1 Цель работы и постановка задачи 3
1.1 Цель работы 3
1.2 Постановка задачи 3
1.3 Вариативная часть задания 3
2 Реализация 4
2.1 Содержание файла main.cpp 4
2.2 Содержание файла function.cpp 4
2.3 Содержание файла header.h 26
2.4 Содержание файла … ..... **Ошибка! Закладка не определена.**
3 Демонстрация работы приложения 28

1 Цель работы и постановка задачи

1.1 Цель работы

Научиться взаимодействовать со структурированными текстовыми файлами различных форматов, использовать сериализацию и десериализацию для сохранения и загрузки информации.

1.2 Постановка задачи

Необходимо разработать приложение с графическим пользовательским интерфейсом согласно индивидуальному варианту. Для разработки можно использовать фреймворк Qt или язык программирования C# совместно с технологиями Windows Forms или WPF. По согласованию с преподавателем возможно использование фреймворка Xamarin или технологии MAUI. В ходе работы приложение должно хранить свои данные в файле формата XML или JSON и обращаться к ним по мере необходимости.

1.3 Вариативная часть задания

Вариант №10.

Список студентов для людей с аллергией на БД. В файле хранятся студенты, поделённые на группы, поделённые по направлениям подготовки. Для каждого студента хранится его ФИО,и номер зачётки. На форме представлены три QListWidget и текстовое поле. Первый хранит список направлений подготовки. После выбора направления, во втором появляются группы, относящиеся к нему. После выбора группы, в третьем списке появляются ФИО студентов из этой группы. Выбор студента приводит к отображению в текстовом поле его номера зачётки. Также должна быть реализована возможность добавления нового студента, группы и направления подготовки.

2 Реализация

Файл с функционалом, main для сборки, header с прототипами, работа выполнена без .ui файла.

2.1 Содержание файла main.cpp

```
#include "mainwindow.h"
#include <QApplication>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

2.2 Содержание файла function.cpp

```
#include "mainwindow.h"
MainWindow::MainWindow(QWidget *parent)
    : QWidget(parent)
{
    // Create layout
    QVBoxLayout *supportLayout = new QVBoxLayout();
    QHBoxLayout *mainLayout = new QHBoxLayout();

    // Create objects (use class members)
    directionsList = new QListWidget;
    groupsList = new QListWidget;
    studentsList = new QListWidget;

    // Create labels
    QLabel *directionsLabel = new QLabel("Направление");
    QLabel *groupsLabel = new QLabel("Группы");
```

```
    QLabel *studentsLabel = new QLabel("Студенты");

    // Add labels and objects to layout
    QVBoxLayout *directionsLayout = new QVBoxLayout();
    directionsLayout->addWidget(directionsLabel);
    directionsLayout->addWidget(directionsList);

    QVBoxLayout *groupsLayout = new QVBoxLayout();
    groupsLayout->addWidget(groupsLabel);
    groupsLayout->addWidget(groupsList);

    QVBoxLayout *studentsLayout = new QVBoxLayout();
    studentsLayout->addWidget(studentsLabel);
    studentsLayout->addWidget(studentsList);

    mainLayout->addLayout(directionsLayout);
    mainLayout->addLayout(groupsLayout);
    mainLayout->addLayout(studentsLayout);

    // Add buttons
    QPushButton *addButton = new QPushButton("Добавить");
    QPushButton       *addDirectionButton       =       new
QPushButton("Добавить направление");
    QPushButton *removeButton = new QPushButton("Удалить");
    QPushButton *exitButton = new QPushButton("Выйти");

    supportLayout->addStretch();
    supportLayout->addWidget(addButton);
    supportLayout->addWidget(addDirectionButton);
```

```
    supportLayout->addWidget(removeButton);
    supportLayout->addWidget(exitButton);

    supportLayout->setContentsMargins(10, 10, 10, 10);

    mainLayout->addLayout(supportLayout);

    // Connect signals and slots
    connect(addButton,      &QPushButton::clicked,      this,
&MainWindow::addButtonClicked);
    connect(addDirectionButton, &QPushButton::clicked, this,
&MainWindow::addDirection);
    connect(removeButton,    &QPushButton::clicked,    this,
&MainWindow::removeButtonClicked);
    connect(exitButton,     &QPushButton::clicked,     this,
&MainWindow::close);

    mainLayout->setStretch(0, 1);
    mainLayout->setStretch(1, 1);
    mainLayout->setStretch(2, 1);

    loadJsonData();

    groupsList->clear();
    studentsList->clear();

    connect(directionsList,  &QListWidget::currentRowChanged,
this, &MainWindow::onDirectionChanged);
```

```
    connect(groupsList,      &QListWidget::currentRowChanged,
this, &MainWindow::onGroupChanged);
    connect(studentsList,  &QListWidget::itemClicked,  this,
&MainWindow::onStudentClicked);


    setLayout(mainLayout);
    setWindowTitle("База данных");
}

void MainWindow::loadJsonData()
{
    QFile
file("/Users/nikitamakhov/Documents/qt/QtTrainPr3/MainWindow
/base.json");
    if (!file.open(QIODevice::ReadOnly)) {
        QMessageBox::warning(this,  "Ошибка",  "Не  удалось
открыть файл");
        return;
    }

    QByteArray jsonData = file.readAll(); // jsonData - это
QByteArray
    file.close();

    QJsonDocument                jsonDoc                =
QJsonDocument::fromJson(jsonData);   //   Преобразование   в
QJsonDocument
```

```
    QJsonObject jsonObject = jsonDoc.object(); // Получение объекта из документа
    this->jsonData = jsonObject; // Сохраняем jsonObject в поле класса

    QJsonArray directions = jsonObject["directions"].toArray(); // Теперь работаем с QJsonObject

    // Populate directions list
    for (const QJsonValue &directionValue : directions) {
        QJsonObject direction = directionValue.toObject();
        QString directionName = direction.value("name").toString();
        directionsList->addItem(directionName);
    }
}

void MainWindow::onDirectionChanged(int row)
{
    if (row < 0 || row >= directionsList->count()) {
        return;
    }

    groupsList->clear();
    studentsList->clear();

    // Get selected direction
```

```
    QString directionName = directionsList->item(row)->text();

    QJsonArray directions = jsonData["directions"].toArray();
// Теперь используем jsonData
    for (const QJsonValue &directionValue : directions) {
        QJsonObject direction = directionValue.toObject();
        if (direction.value("name").toString() == directionName) {
            // Populate groups list
            QJsonArray groups = direction.value("groups").toArray();
            for (const QJsonValue &groupValue : groups) {
                QJsonObject group = groupValue.toObject();
                QString groupName = group.value("name").toString();
                groupsList->addItem(groupName);
            }
            break;
        }
    }
}

void MainWindow::onGroupChanged(int row)
{
    if (row < 0 || row >= groupsList->count()) {
        return;
    }
```

```
    studentsList->clear();

    // Get selected group
    QString groupName = groupsList->item(row)->text();

    QJsonArray directions = jsonData["directions"].toArray();
// Теперь используем jsonData
    for (const QJsonValue &directionValue : directions) {
        QJsonObject direction = directionValue.toObject();
        QJsonArray groups =
direction.value("groups").toArray();
        for (const QJsonValue &groupValue : groups) {
            QJsonObject group = groupValue.toObject();
            if (group.value("name").toString() == groupName)
{
                // Populate students list
                QJsonArray students =
group.value("students").toArray();
                for (const QJsonValue &studentValue :
students) {
                    QJsonObject student =
studentValue.toObject();
                    QString studentName =
student.value("full_name").toString();
                    studentsList->addItem(studentName);
                }
                break;
            }
        }
```

```
    }
}

void MainWindow::onStudentClicked(QListWidgetItem *item)
{
    QString studentName = item->text();

    QJsonArray directions = jsonData["directions"].toArray();
// Теперь используем jsonData
    for (const QJsonValue &directionValue : directions) {
        QJsonObject direction = directionValue.toObject();
        QJsonArray groups =
direction.value("groups").toArray();
        for (const QJsonValue &groupValue : groups) {
            QJsonObject group = groupValue.toObject();
            QJsonArray students =
group.value("students").toArray();
            for (const QJsonValue &studentValue : students) {
                QJsonObject student =
studentValue.toObject();
                if (student.value("full_name").toString() ==
studentName) {
                    QString studentId =
student.value("student_id").toString();
                    QMessageBox::information(this,
"Информация", "Номер зачетки: " + studentId);
                    return;
                }
            }
```

```
        }
    }
}

void MainWindow::addStudent() // Реализация метода добавления студента
{
    bool ok;
    QString studentName = QInputDialog::getText(this, "Добавить студента", "Имя студента:", QLineEdit::Normal, "", &ok);
    if (!ok || studentName.isEmpty()) {
        return; // Отмена или пустой ввод
    }

    QString studentId = QInputDialog::getText(this, "Добавить студента", "Номер зачетки:", QLineEdit::Normal, "", &ok);
    if (!ok || studentId.isEmpty()) {
        return; // Отмена или пустой ввод
    }

    int groupRow = groupsList->currentRow();
    if (groupRow < 0 || groupRow >= groupsList->count()) {
        QMessageBox::warning(this, "Ошибка", "Сначала выберите группу.");
        return;
    }

    QString groupName = groupsList->item(groupRow)->text();
```

```
    QJsonArray directions = jsonData["directions"].toArray();

    for (int i = 0; i < directions.size(); ++i) {
        QJsonObject direction = directions[i].toObject();
        QJsonArray groups = direction.value("groups").toArray();
        for (int j = 0; j < groups.size(); ++j) {
            QJsonObject group = groups[j].toObject();
            if (group.value("name").toString() == groupName) {
                // Создаём новый объект студента
                QJsonObject newStudent;
                newStudent["full_name"] = studentName;
                newStudent["student_id"] = studentId;

                // Добавляем нового студента в массив студентов
                QJsonArray students = group.value("students").toArray();
                students.append(newStudent);
                group["students"] = students; // Обновляем группу с модифицированными студентами
                groups[j] = group; // Обновляем значение группы

                // Обновляем JSON объект
                direction["groups"] = groups;
                directions[i] = direction; // Обновляем значение направления
```

```
                jsonData["directions"] = directions;

                // Сохранение изменений в JSON файл
                QFile
file("/Users/nikitamakhov/Documents/qt/QtTrainPr3/MainWindow
/base.json");
                if (!file.open(QIODevice::WriteOnly)) {
                    QMessageBox::warning(this, "Ошибка", "Не
удалось открыть файл для записи.");
                    return;
                }
                QJsonDocument updatedDoc(jsonData);
                file.write(updatedDoc.toJson());
                file.close();

                // Обновляем список студентов
                studentsList->addItem(studentName);
                QMessageBox::information(this,      "Успех",
"Студент добавлен успешно!");
                return;
            }
        }
    }
}
void MainWindow::addGroup() {
    bool ok;
    QString groupName = QInputDialog::getText(this, "Добавить
группу", "Имя группы:", QLineEdit::Normal, "", &ok);
    if (!ok || groupName.isEmpty()) {
```

```
        return; // Отмена или пустой ввод
    }

    int directionRow = directionsList->currentRow();
    if (directionRow < 0 || directionRow >= directionsList->count()) {
        QMessageBox::warning(this, "Ошибка", "Сначала выберите направление.");
        return;
    }

    QString directionName = directionsList->item(directionRow)->text();
    QJsonArray directions = jsonData["directions"].toArray();

    for (int i = 0; i < directions.size(); ++i) {
        QJsonObject direction = directions[i].toObject();
        if (direction.value("name").toString() == directionName) {
            // Создаем новый объект группы
            QJsonObject newGroup;
            newGroup["name"] = groupName;
            newGroup["students"] = QJsonArray(); // Инициализируем пустой массив студентов

            // Добавляем группу в массив групп
            QJsonArray groups = direction.value("groups").toArray();
            groups.append(newGroup);
```

```
            direction["groups"] = groups; // Обновляем
направление
            directions[i] = direction; // Обновляем массив
направлений
            jsonData["directions"] = directions;

            // Сохранение изменений в JSON файл
            QFile
file("/Users/nikitamakhov/Documents/qt/QtTrainPr3/MainWindow
/base.json");
            if (!file.open(QIODevice::WriteOnly)) {
                QMessageBox::warning(this, "Ошибка", "Не
удалось открыть файл для записи.");
                return;
            }
            QJsonDocument updatedDoc(jsonData);
            file.write(updatedDoc.toJson());
            file.close();

            // Обновляем список групп
            groupsList->addItem(groupName);
            QMessageBox::information(this, "Успех", "Группа
добавлена успешно!");
            return;
        }
    }
}
```

```
void MainWindow::removeStudent() // Реализация метода удаления студента
{
    int studentRow = studentsList->currentRow();
    if (studentRow < 0 || studentRow >= studentsList->count()) {
        QMessageBox::warning(this, "Ошибка", "Сначала выберите студента.");
        return;
    }

    QString studentName = studentsList->item(studentRow)->text();
    int groupRow = groupsList->currentRow();
    if (groupRow < 0 || groupRow >= groupsList->count()) {
        QMessageBox::warning(this, "Ошибка", "Сначала выберите группу.");
        return;
    }

    QString groupName = groupsList->item(groupRow)->text();
    QJsonArray directions = jsonData["directions"].toArray();

    for (int i = 0; i < directions.size(); ++i) {
        QJsonObject direction = directions[i].toObject();
        QJsonArray groups = direction.value("groups").toArray();
        for (int j = 0; j < groups.size(); ++j) {
            QJsonObject group = groups[j].toObject();
```

```
            if (group.value("name").toString() == groupName)
{
                // Получаем массив студентов
                QJsonArray             students             =
group.value("students").toArray();
                for (int k = 0; k < students.size(); ++k) {
                    QJsonObject           student           =
students[k].toObject();
                    if (student.value("full_name").toString()
== studentName) {
                        // Удаляем студента из массива
                        students.removeAt(k);
                        group["students"]   =   students;   //
Обновляем группу
                        groups[j]   =   group;   //   Обновляем
значение группы

                        // Обновляем JSON объект
                        direction["groups"] = groups;
                        directions[i]    =    direction;    //
Обновляем значение направления
                        jsonData["directions"] = directions;

                        // Сохранение изменений в JSON файл
                        QFile
file("/Users/nikitamakhov/Documents/qt/QtTrainPr3/MainWindow
/base.json");
                        if (!file.open(QIODevice::WriteOnly))
{
```

```
                            QMessageBox::warning(this,
"Ошибка", "Не удалось открыть файл для записи.");
                            return;
                        }
                        QJsonDocument updatedDoc(jsonData);
                        file.write(updatedDoc.toJson());
                        file.close();

                        // Обновляем список студентов
                        delete                  studentsList-
>takeItem(studentRow); // Удаляем студента из списка
                        QMessageBox::information(this,
"Успех", "Студент удален успешно!");
                        return;
                    }
                }
            }
        }
    }
}
void MainWindow::removeGroup() {
    // Get the selected group index
    int groupRow = groupsList->currentRow();
    if (groupRow < 0 || groupRow >= groupsList->count()) {
        QMessageBox::warning(this,    "Ошибка",    "Сначала
выберите группу.");
        return;
    }
```

```
    QString groupName = groupsList->item(groupRow)->text();

    // Get the selected direction index
    int directionRow = directionsList->currentRow();
    if (directionRow < 0 || directionRow >= directionsList->count()) {
        QMessageBox::warning(this, "Ошибка", "Сначала выберите направление.");
        return;
    }

    QString directionName = directionsList->item(directionRow)->text();
    QJsonArray directions = jsonData["directions"].toArray();

    // Iterate through the directions
    for (int i = 0; i < directions.size(); ++i) {
        QJsonObject direction = directions[i].toObject();
        if (direction.value("name").toString() == directionName) {
            QJsonArray groups = direction.value("groups").toArray();

            // Find and remove the group
            for (int j = 0; j < groups.size(); ++j) {
                QJsonObject group = groups[j].toObject();
                if (group.value("name").toString() == groupName) {
                    // Remove the group from the array
```

```
                    groups.removeAt(j);
                    direction["groups"] = groups; // Update
direction with the modified groups
                    directions[i] = direction; // Update
directions array
                    jsonData["directions"] = directions; //
Update jsonData

                    // Save changes to the JSON file
                    QFile
file("/Users/nikitamakhov/Documents/qt/QtTrainPr3/MainWindow
/base.json");
                    if (!file.open(QIODevice::WriteOnly)) {
                        QMessageBox::warning(this, "Ошибка",
"Не удалось открыть файл для записи.");
                        return;
                    }
                    QJsonDocument updatedDoc(jsonData);
                    file.write(updatedDoc.toJson());
                    file.close();

                    // Remove the group from the UI
                    delete groupsList->takeItem(groupRow); //
Remove the group from the list widget
                    QMessageBox::information(this, "Успех",
"Группа удалена успешно!");
                    return;
                }
            }
```

```
        }
    }

    // If we reach here, the group was not found (optional)
    QMessageBox::warning(this, "Ошибка", "Группа не найдена.");
}

void MainWindow::addButtonClicked() {
    if (groupsList->currentRow() >= 0) { // Если выбрана группа, добавляем студента
        addStudent();
    } else if (directionsList->currentRow() >= 0) { // Если выбрано направление, добавляем группу
        addGroup();
    } else {
        QMessageBox::warning(this, "Ошибка", "Сначала выберите направление или группу.");
    }
}

void MainWindow::removeButtonClicked() {
    int studentRow = studentsList->currentRow();
    if (studentRow >= 0) {
        removeStudent();
        return;
    }

    int groupRow = groupsList->currentRow();
```

```
    if (groupRow >= 0) {
        removeGroup();
        return;
    }

    int directionRow = directionsList->currentRow();
    if (directionRow >= 0) {
        removeDirection();
        return;
    }

    QMessageBox::warning(this, "Ошибка", "Сначала выберите студента, группу или направление для удаления.");
}


void MainWindow::addDirection() {
    bool ok;
    QString directionName = QInputDialog::getText(this, "Добавить направление", "Имя направления:", QLineEdit::Normal, "", &ok);
    if (!ok || directionName.isEmpty()) {
        return; // Отмена или пустой ввод
    }

    QJsonObject newDirection;
    newDirection["name"] = directionName;
    newDirection["groups"] = QJsonArray();
```

```
    QJsonArray directions = jsonData["directions"].toArray();
    directions.append(newDirection);
    jsonData["directions"] = directions;


    QFile
file("/Users/nikitamakhov/Documents/qt/QtTrainPr3/MainWindow
/base.json");
    if (!file.open(QIODevice::WriteOnly)) {
        QMessageBox::warning(this,  "Ошибка",  "Не  удалось
открыть файл для записи.");
        return;
    }
    QJsonDocument updatedDoc(jsonData);
    file.write(updatedDoc.toJson());
    file.close();

    directionsList->addItem(directionName);
    QMessageBox::information(this,  "Успех",  "Направление
добавлено успешно!");
}
void MainWindow::removeDirection() {
    // Get the selected direction index
    int directionRow = directionsList->currentRow();
    if (directionRow < 0 || directionRow >= directionsList-
>count()) {
        QMessageBox::warning(this,    "Ошибка",    "Сначала
выберите направление.");
        return;
```

```
    }

    QString directionName = directionsList->item(directionRow)->text();
    QJsonArray directions = jsonData["directions"].toArray();

    for (int i = 0; i < directions.size(); ++i) {
        QJsonObject direction = directions[i].toObject();
        if (direction.value("name").toString() == directionName) {

            directions.removeAt(i);
            jsonData["directions"] = directions;

            QFile file("/Users/nikitamakhov/Documents/qt/QtTrainPr3/MainWindow/base.json");
            if (!file.open(QIODevice::WriteOnly)) {
                QMessageBox::warning(this, "Ошибка", "Не удалось открыть файл для записи.");
                return;
            }
            QJsonDocument updatedDoc(jsonData);
            file.write(updatedDoc.toJson());
            file.close();

            delete directionsList->takeItem(directionRow);
            QMessageBox::information(this, "Успех", "Направление удалено успешно!");
```

```
            return;
        }
    }

    QMessageBox::warning(this, "Ошибка", "Направление не
найдено.");
}

MainWindow::~MainWindow() {}
```

2.3 Содержание файла header.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QWidget>
#include <QListWidget>
#include <QJsonDocument>
#include <QJsonObject>
#include <QJsonArray>
#include <QPushButton>
#include <QMessageBox>
#include <QInputDialog>
#include <QFile>
#include <QHBoxLayout>
#include <QLabel>

class MainWindow : public QWidget
{
    Q_OBJECT
```

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

private slots:
    void loadJsonData();
    void onDirectionChanged(int row);
    void onGroupChanged(int row);
    void addButtonClicked();
    void onStudentClicked(QListWidgetItem *item);
    void addStudent();
    void addGroup();
    void addDirection();
    void removeStudent();
    void removeGroup();
    void removeDirection();
    void removeButtonClicked();

private:
    QListWidget *directionsList;
    QListWidget *groupsList;
    QListWidget *studentsList;
    QJsonObject jsonData; // Объект для хранения данных JSON
};

#endif // MAINWINDOW_H
```

3 Демонстрация работы приложения

Программа собралась. Результат представлен на рисунке 1.

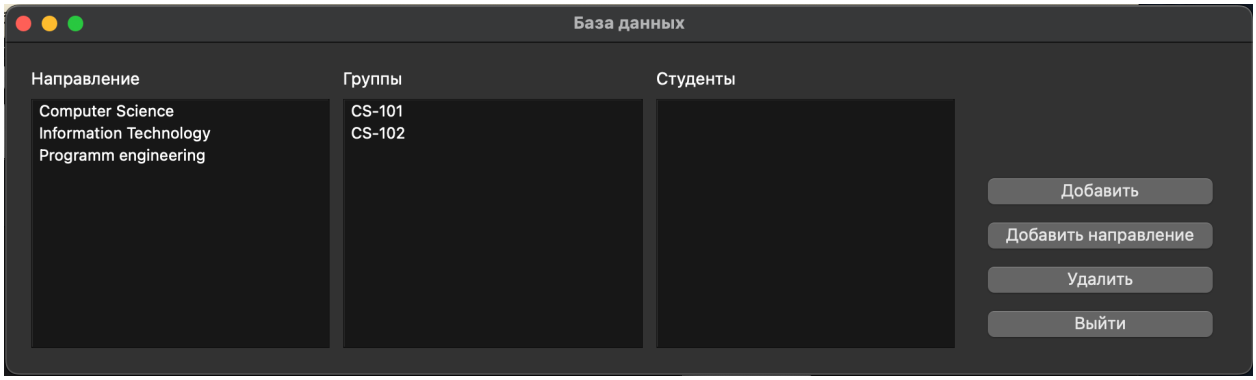

Рисунок 1 — Запуск программы, список направлений

Запуск программы, пользователя встречает окно с тремя QListWidget, выбрав направление, пользователю выводится список группу по данному направлению. Результат представлен на рисунке 2.

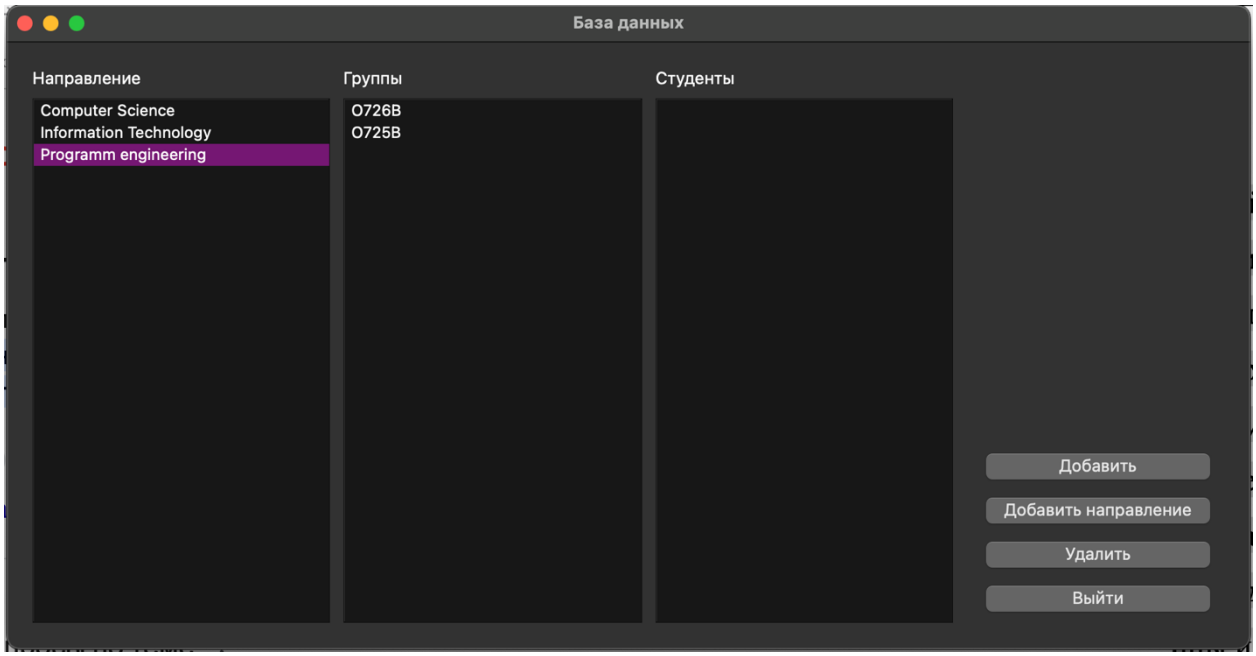

Рисунок 2 — Запуск программы, список направлений, список групп по выбранному направлению

Выбор группы, происходит вывод списка студентов. Результат представлен на рисунке 3.

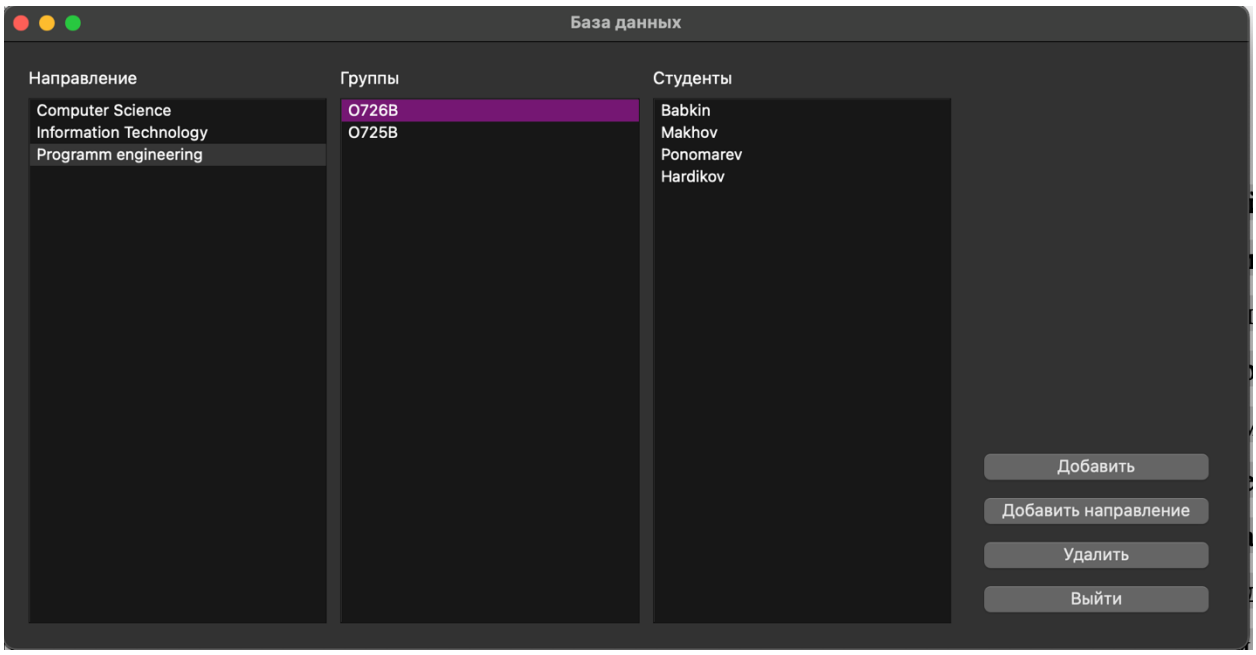

Рисунок 3 —

Добавим студента, можно сделать во вкладке студентов, по нажатию на кнопку Добавить. Результат представлен на рисунке 4.

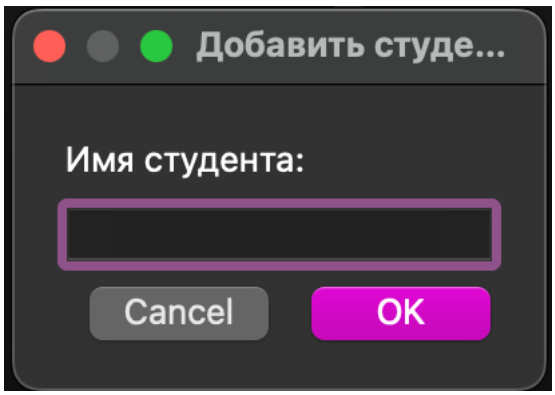

Рисунок 4 — Ввод ФИО студента

После ввода имени, программа предлагает ввести номер зачетной книжки. Результат представлен на рисунке 5.

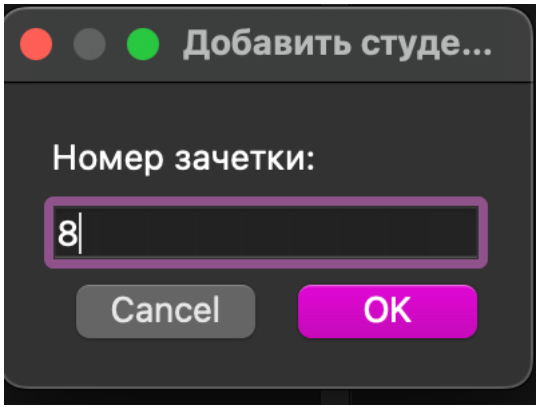

Рисунок 5 — Ввод номера зачетной книжки

Данные о студенте успешно добавлены. Результат представлен на рисунке 6.

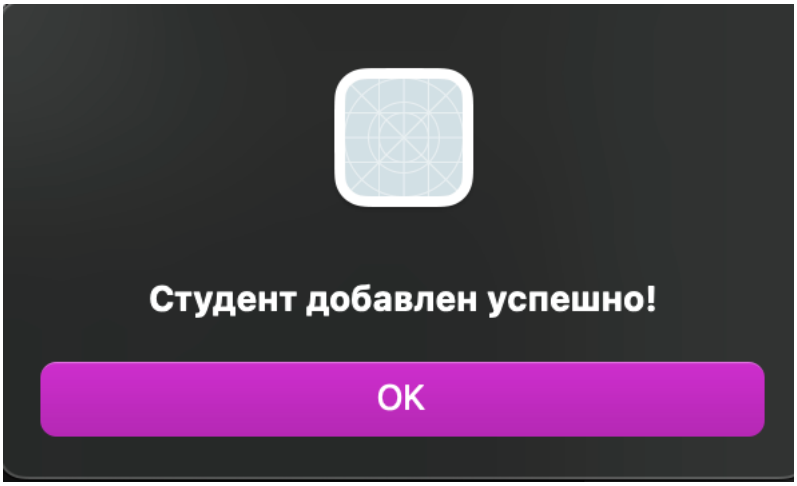

Рисунок 6 — Данные о студенте успешно добавлены

Новый студент появился в списке студентов, значит он был занесен в JSON файл. Результат представлен на рисунке 7.

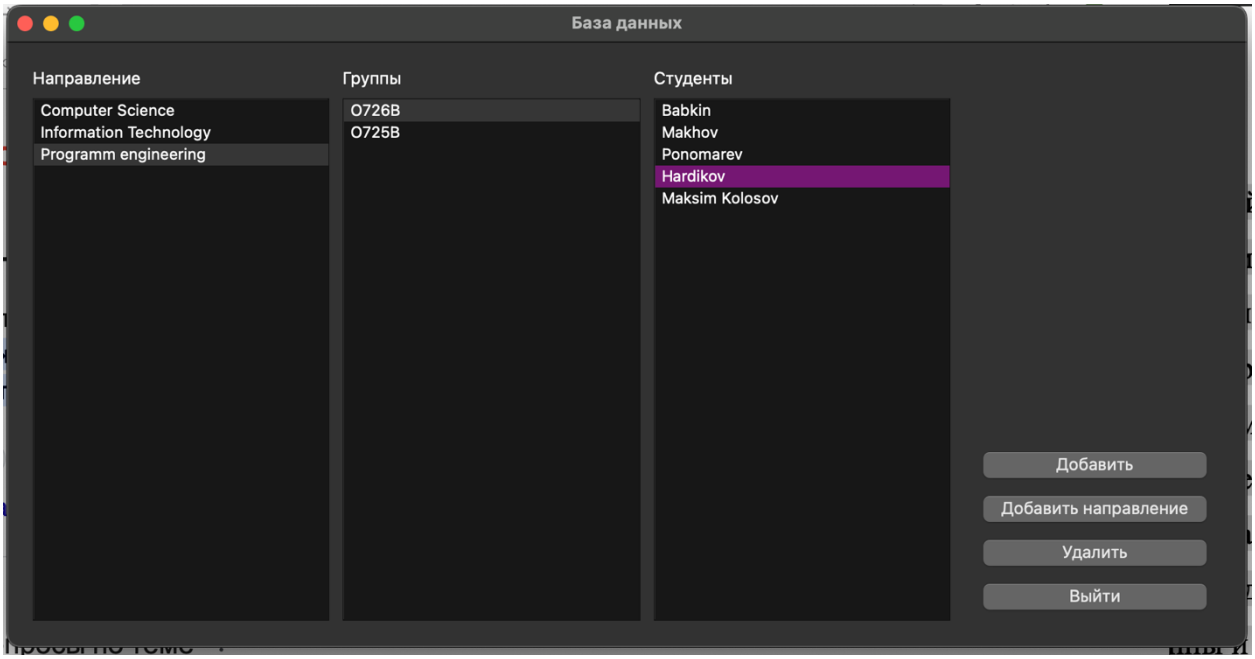

Рисунок 7 — Новый студент

Нажав по ФИО студента, выводится номер его зачетной книжки. Результат представлен на рисунке 8.

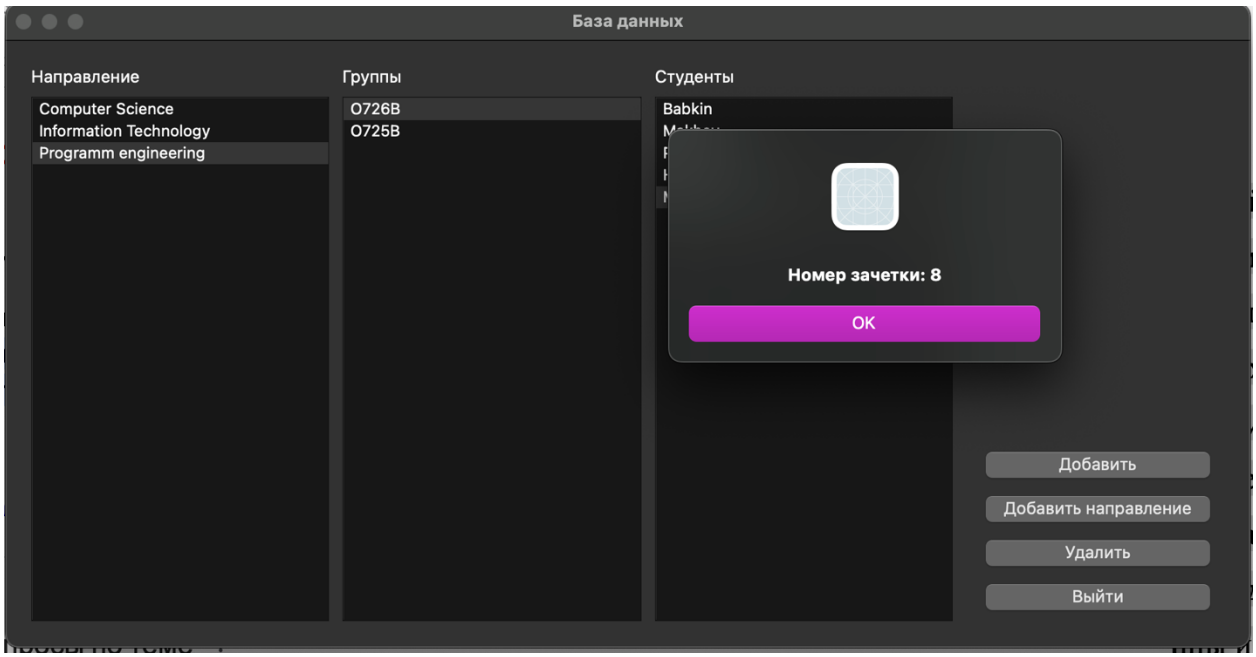

Рисунок 8 — Вывод номера зачетной книжки

Удалим студента из списка учащихся. Результат представлен на рисунке 9.

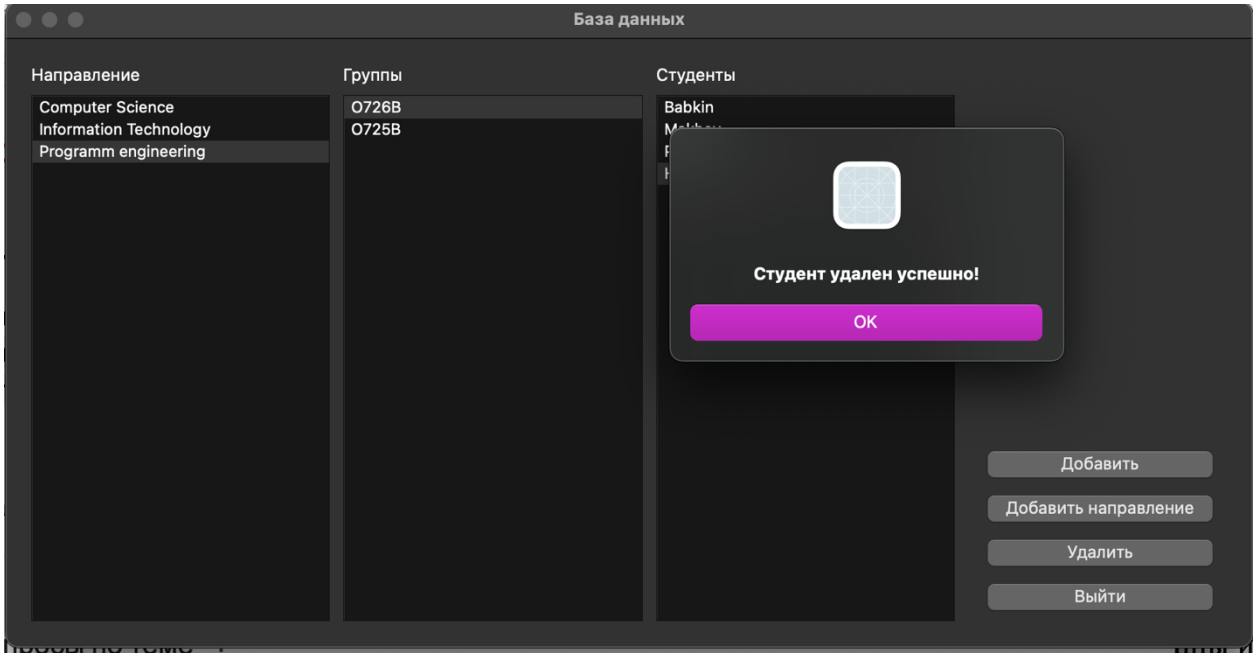

Рисунок 9 — Удаление студента из списка учащихся

Удалим группу, для этого перейдем в другую с заранее заготовленным студентами под отчисление. Результат представлен на рисунке 10.

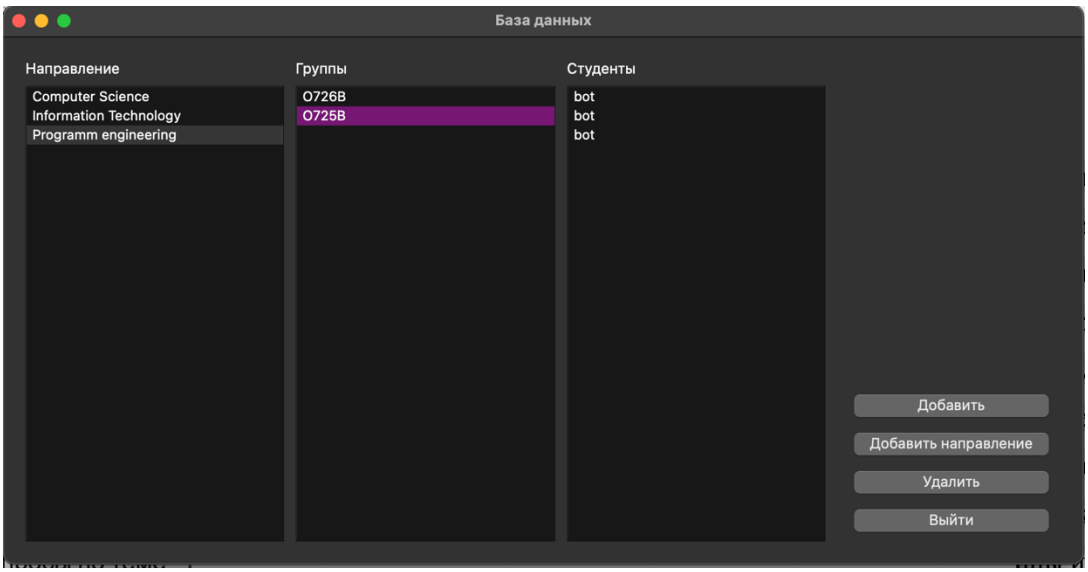

Рисунок 10 — Просмотр группе, удаление группы

Группа успешна удалена. Результат представлен на рисунке 11.

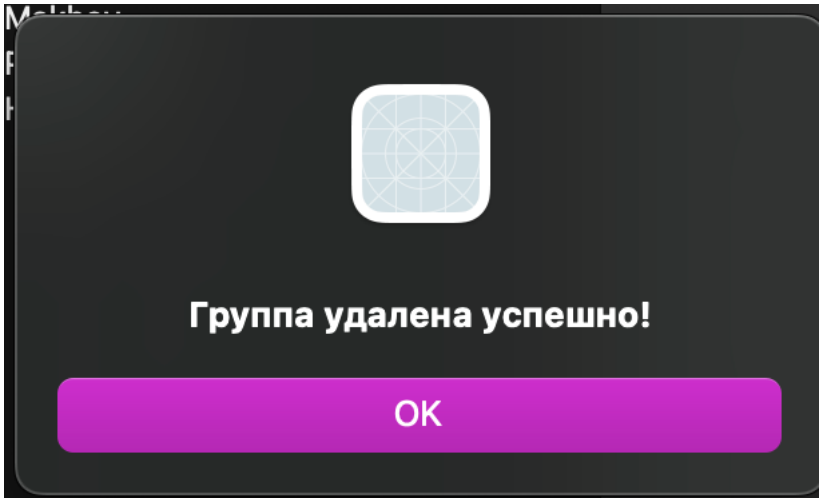

Рисунок 11 — Группа успешна удалена

Создадим группу. Результат представлен на рисунке 12.

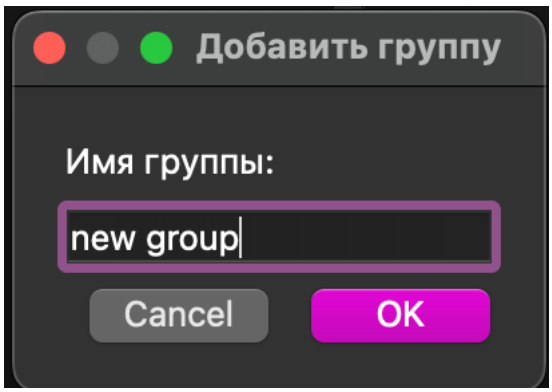

Рисунок 12 — Создание группы

Добавим новое направление, по кнопке добавить направление. Результат представлен на рисунке 13.

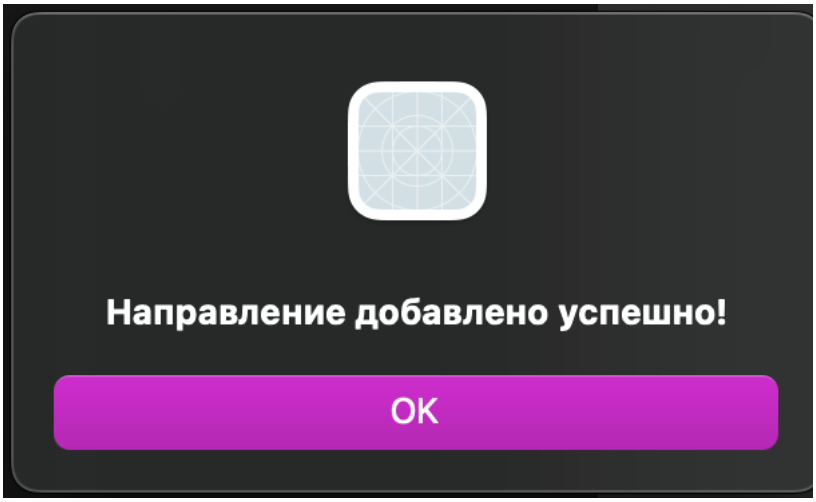

Рисунок 13 — Добавление нового направления

Удалим направление. Результат представлен на рисунке 14.

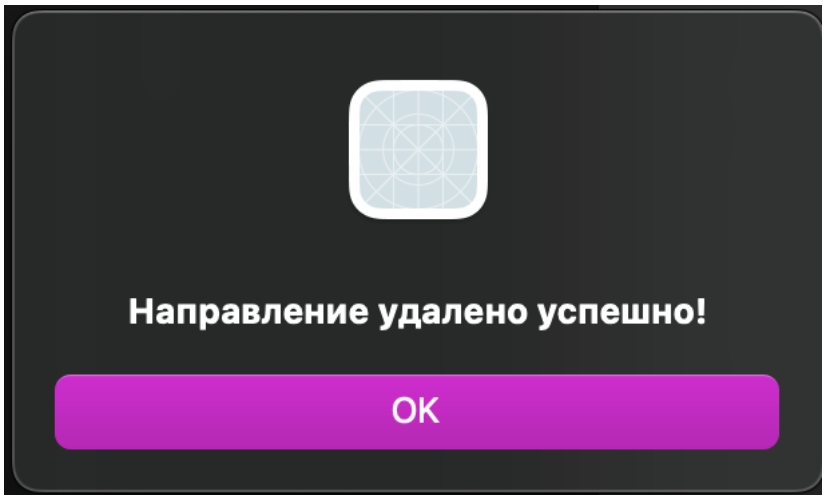

Рисунок 14 — Удаление направления