МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Нижегородский государственный университет им. Н.И. Лобачевского»
Национальный исследовательский университет

Институт информационных технологий, математики и механики
Кафедра программной инженерии

ОТЧЕТ

по дисциплине «Разработка мобильных приложений»

Индивидуальное практическое задание:

«Перевод изображения в оттенки серого»

Выполнил:
студент группы 381706-1
Денисов В. Л.

Проверил:
доцент кафедры программной инженерии
Борисов Н. А.

Нижний Новгород
2020.

Содержание

1 Цели

Целью данной лабораторной работы является создание приложения, позволяющего выполнить фильтрацию изображения по переводу в оттенки серого.

Постановка задачи

1. Создать класс-фильтр, предоставляющий необходимый функционал для работы с изображением, которое требуется обработать. Класс должен содержать следующие методы:

   1.1.Получения изображения из интернета по указанной ссылке.

   1.2.Перевода изображения в оттенки серого.

2. Создать пользовательский интерфейс с использованием элементов, предоставляемых *SailfishOS*, который позволит продемонстрировать корректную работу приложения. В интерфейс должны входить: поле для ввода ссылки до изображения; кнопка, инициирующая загрузку изображения; 2 поля для изображений –для исходного и обработанного соответственно.

3 Решение поставленных задач

Создадим проект со стандартной заготовкой приложения, где файлом главной страницы приложения будет являться *FirstPage.qml*.

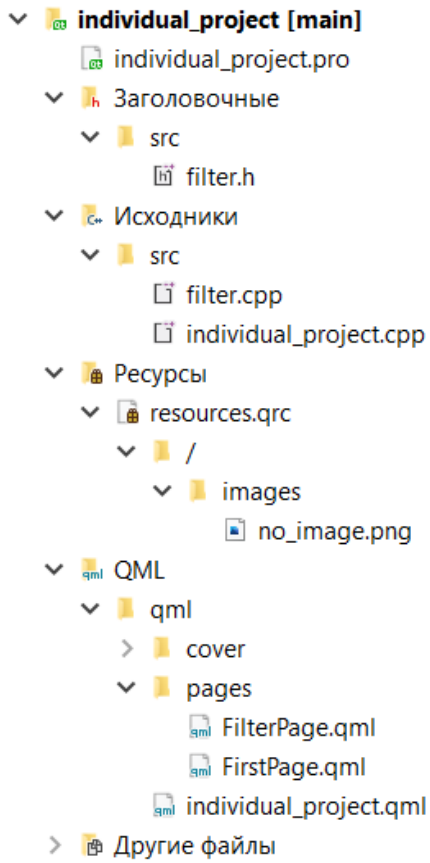

Рисунок 1 Структура проекта.

Демонстрация работы фильтра будет представлена на отдельной странице – *FilterPage.qml*, переход на которую выполняется путем нажатия соответствующей кнопки, расположенной на *FirstPage*. Демонстрационная страница будет отправлена в *pageStack*.

3.1 Создание класса фильтра и описание его функционала

Класс-фильтр опишем в заголовочном файле *filter.h*. Реализация класса будет представлена в файле *filter.cpp*. Для дальнейшего использования класса зарегистрируем его в приложении путем вызова метода *qmlRegisterType<Filter>("custom.Filter", 1, 0, "Filter*") в *main()*-функции приложения.

Публичные методы, которые имеет класс-фильтр:

- *getUrl()* – получить, хранимую ссылку по которой загружается изображение;

- *setUrl(QString)* – установить ссылку, по которой расположено загружаемое изображение;
- *loadImage()* – загрузить изображение по хранимой ссылке в объект класса;
- *QString grayfilter()* – выполнить перевод хранимого изображения в оттенки серого и вернуть результат.

Защищенный метод:

- *QString reciveImage()* – преобразователь обработанного изображения в путь до него, который может быть использован контейнером *Image*, предоставляемым *Sailfish*, для отображения на экране.

Слот:

- *replyFinish* – обработчик, который вызывается по завершению загрузки изображения из интернета при помощи *QNetworkAccessManager*. Сохраняет полученный из интернета ответ на запрос загрузки, в случае, если не произошло ошибок.

3.2 Перевод изображения в оттенки серого

Непосредственно перевод в оттенки серого выполняется путем полного прохода по изображению по высоте и ширине и обработки каждого пикселя по отдельности.

В переменную *newColor*, имеющую тип *QColor*, записывается значение пикселя путем вызова метода *image.pixelColor(j, i)*, где *j* и *i* его координаты, а *image* – изображение, хранимое при помощи *QImage*. Затем получаются значения цвета по каждому каналу *R*, *G*, *B* при помощи вызова методов *newColor.red()*, *newColor.green()*, *newColor.blue()* соответственно.

Значение цвета пикселя в оттенках серого вычисляется по следующей формуле:

$$gray = 0.2952 * R + 0.5547 * \mathrm{G} + 0.148 * \mathrm{B}$$

Полученное значение gray устанавливается в каждый канал пикселя вызовом методов *newColor.setRed(gray)*, *newColor.setGreen(gray)*, *newColor.setBlue(gray)*. Обновленный пиксель возвращается на изображение – *image.setPixelColor(j, i, newColor)*.

3.3 Демонстрация работы фильтра

Использование класса-фильтра непосредственно на странице в приложении осуществляется следующим образом. Импортируем зарегистрированный функционал *import custom.Filter 1.0*. Создаем элемент *Filter (id: filter).*

Помимо этого, создаем еще ряд элементов:

- Текстовое поле *TextField*, (*id*: *txtfield*), используемое для ввода ссылки до изображения;
- Кнопку *Button* (*id: getUrlButton*) для установки значения ссылки, указанной в *txtfield,* в объект класса *Filter* и последующей загрузки изображения;
- Кнопку *Button* (*id: grayButton*) для запуска фильтра по переводу изображения в оттенки серого;
- 2 текстовых метки *Label* для подписей исходного и обработанного изображения, а также 2 контейнера для этих изображений – *Image*.

В результате сможем наблюдать следующую картину:

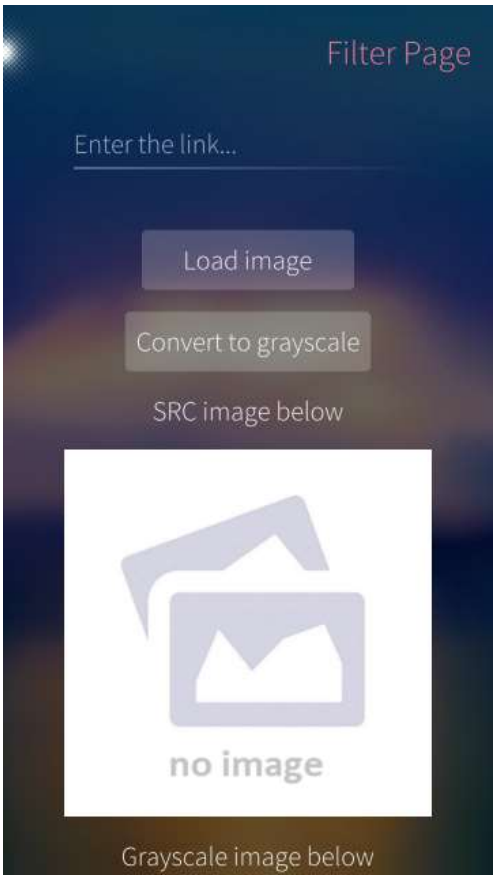

Рисунок 2 Изображение не загружено.

Рисунок 3 Указана ссылка до изображения и произведена загрузка.

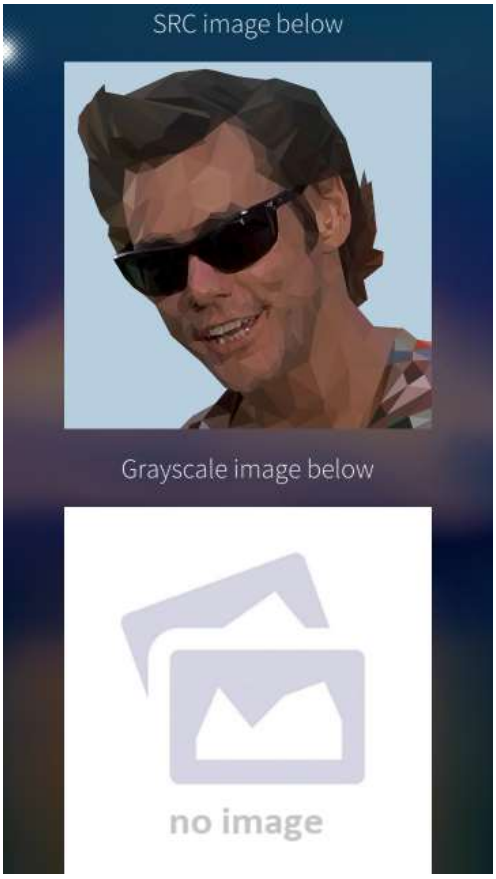

Рисунок 4 Нажатие на кнопку запуска фильтрации не произведено.

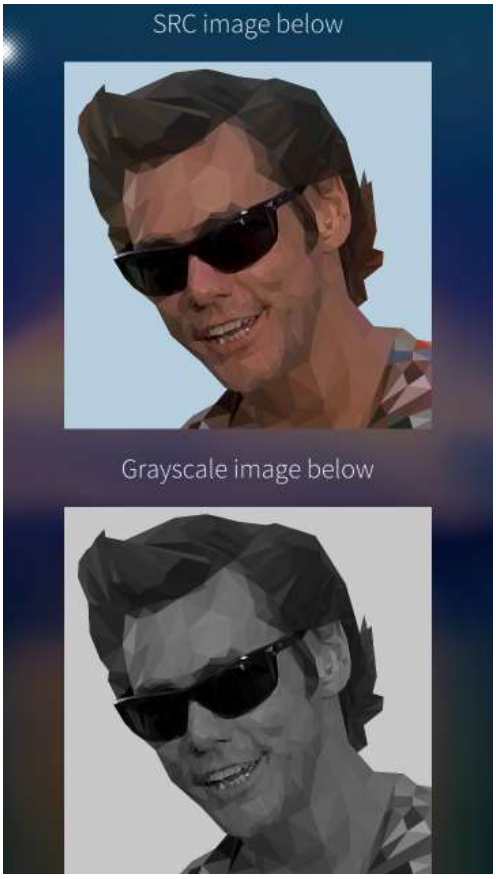

Рисунок 5 Выполнен перевод изображения в оттенки серого.

4 Приложение

4.1 Файл FirstPage.qml

```
import QtQuick 2.0
import Sailfish.Silica 1.0

Page {
    id: page
    allowedOrientations: Orientation.All

    SilicaFlickable {
        anchors.fill: parent
        contentHeight: column.height

        Column {
            id: column
            width: page.width
            spacing: Theme.paddingLarge
            PageHeader {
                title: qsTr("Main Page")
            }
            Button {
                text: qsTr("Filter-demo")
                onClicked: pageStack.push(Qt.resolvedUrl("FilterPage.qml"))
                anchors.horizontalCenter: parent.horizontalCenter
            }
        } // Column
    } // SilicaFlickable
} // Page
```

4.2 Файл FilterPage.qml

```
import QtQuick 2.0
import Sailfish.Silica 1.0
import custom.Filter 1.0


Page {
    id: page

    allowedOrientations: Orientation.All

    Filter {
        id: filter
    }
```

```
SilicaFlickable {
    anchors.fill: parent
    contentHeight: column.height

    Column {
        id: column
        width: page.width
        spacing: Theme.paddingLarge
        PageHeader {
            title: qsTr("Filter Page")
        }

        // Setting a link to an image
        TextField {
            id: txtfield
            width: parent.width / 5 * 4
            placeholderText: "Enter the link..."
            anchors.horizontalCenter: parent.horizontalCenter
        }
        Button {
            id: getUrlButton
            text: qsTr("Load image")
            anchors.horizontalCenter: parent.horizontalCenter
            onClicked: {
                if (filter.setUrl(txtfield.text)) {
                    filter.loadImage();
                    image1.source = filter.getUrl();
                    image2.source = "qrc:images/no_image.png";
                }
            }
        }

        // Starting the grayscale filter
        Button{
            text: qsTr("Convert to grayscale")
            anchors.horizontalCenter: parent.horizontalCenter
            onClicked: {
                image2.source = filter.grayfilter();
            }
        }

        Label {
            text: "SRC image below"
            anchors.horizontalCenter: parent.horizontalCenter
        }
        Image {
            id: image1
            width: parent.width / 4 * 3
            height: parent.width / 4 * 3
            anchors.horizontalCenter: parent.horizontalCenter
            source: "qrc:images/no_image.png"
        }

        Label {
            text: "Grayscale image below"
            anchors.horizontalCenter: parent.horizontalCenter
        }
        Image{
            id: image2
            width: parent.width / 4 * 3
            height: parent.width / 4 * 3
```

```
                anchors.horizontalCenter: parent.horizontalCenter
                source: "qrc:images/no_image.png"
            }

        }
    }
}
```

4.3 Файл individual_project.cpp

```
#ifdef QT_QML_DEBUG
#include <QtQuick>
#endif

#include <sailfishapp.h>
#include "filter.h"

int main(int argc, char *argv[])
{
    QGuiApplication *app = SailfishApp::application(argc, argv);
    QQuickView *view = SailfishApp::createView();

    view->setSource(SailfishApp::pathTo("qml/individual_project.qml"));

    qmlRegisterType<Filter>("custom.Filter", 1, 0, "Filter");
    view->show();

    return app->exec();
}
```

4.4 Файл filter.h

```
#ifndef FILTER_H
#define FILTER_H

#include <QObject>
#include <QDebug>
#include <QImage>
#include <QString>
#include <QColor>
#include <QRgb>
//#include <QImageReader>
#include <QByteArray>
#include <QBuffer>
#include <QNetworkAccessManager>
#include <QNetworkRequest>
#include <QNetworkReply>


class Filter : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString url READ getUrl WRITE setUrl NOTIFY onUrlChanged)
private:
    QImage image;
    QString url = "null";
protected:
    QString reciveImage();
public:
    Q_INVOKABLE Filter() : QObject() {}
```

```
    Q_INVOKABLE QString getUrl();
    Q_INVOKABLE bool setUrl(QString _url);
    Q_INVOKABLE void loadImage();
    Q_INVOKABLE QString grayfilter();
signals:
    void onUrlChanged();
private slots:
    void replyFinish(QNetworkReply *reply);
};


#endif // FILTER_H
```

4.5 Файл filter.cpp

```
#include "filter.h"

QString Filter::getUrl()
{
    return url;
}

bool Filter::setUrl(QString _url) {
    bool flag = false;
    flag = QUrl(_url).isValid();
    if (flag) {
        url = _url;
        emit onUrlChanged();
    } else {
        qDebug() << "Invalid URL: " << _url;
    }
    return flag;
}

void Filter::loadImage() {
    QNetworkAccessManager* networkManager = new QNetworkAccessManager(this);

    connect(networkManager,
            SIGNAL(finished(QNetworkReply*)),
            this,
            SLOT(replyFinish(QNetworkReply*)));

    networkManager->get(QNetworkRequest(QUrl(url)));
}

Q_INVOKABLE QString Filter::grayfilter() {
    if (!image.isNull()) {
        qDebug() << "Width and Height: " << image.width() << " " <<
image.height();
        qDebug() << "Processing...";
        for(int i = 0; i < image.height(); ++i) {
            for(int j = 0; j < image.width(); ++j){
                QColor newColor(image.pixelColor(j, i));
                // The calculation of the color in grayscale.
                int grayColorOfPixel = static_cast<int>(0.2952 *
newColor.red() + 0.5547 * newColor.green() + 0.148 * newColor.blue());
                // Setting the color for each channel.
                newColor.setRed(grayColorOfPixel);
                newColor.setGreen(grayColorOfPixel);
                newColor.setBlue(grayColorOfPixel);
                // Converting an image pixel to grayscale
```

```
                image.setPixelColor(j, i, newColor);
            }
        }
        qDebug() <<"Complete";
    } else {
        qDebug() <<"The image is not defined!";
        QString img;
        return img;
    }

    return reciveImage();
}

QString Filter::reciveImage() {
    QByteArray bArray;
    QBuffer buffer(&bArray);
    buffer.open(QIODevice::WriteOnly);
    image.save(&buffer, "JPEG");

    QString newStr("");
    newStr.append(QString::fromLatin1(bArray.toBase64().data()));

    return newStr;
}

// SLOT
void Filter::replyFinish(QNetworkReply *reply) {
    if (reply->error() == QNetworkReply::NoError)
    {
        QByteArray data = reply->readAll();
        image = QImage::fromData(data);
        qDebug() << "Image from data isNull: " << image.isNull();;
    } else {
        qDebug() << "Error QNetworkReply";
    }
}
```

5 Используемая литература

1. Документация QT – https://doc.qt.io/qt-5/qmake-project-files.html