Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Программирование на языках высокого уровня

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

ПРОГРАММА «ФОТОРЕДАКТОР»

БГУИР КП 1-40 02 01 203 ПЗ

Студент: гр. 250502 Болашенко В.С.

Руководитель: Богдан Е. В.

МИНСК 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

C++ – компилируемый язык программирования общего назначения. Он поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование (ООП), обобщенное программирование [1].

Этот язык программирования широко используется для разработки программного обеспечения: создание разнообразных прикладных программ, разработка операционных систем, драйверов устройств, а также видеоигр и многое другое.

Также язык программирование C++ позволяет подключать фреймворки, которые расширяют возможности языка: позволяют создавать оконные приложения, игры, обрабатывать фото- и видеоматериалов и др. Примерами крупных фреймворков являются Qt (для разработки ПО) [2] и OpenCV (обработка изображений, компьютерное зрение) [3].

ООП – подход к программированию как к моделированию информационных объектов, решающий на новом уровне основную задачу структурного программирования: структурирование информации с точки зрения управляемости, что существенно улучшает управляемость самим процессом моделирования, что, в свою очередь, особенно важно при реализации крупных проектов.

Основные принципы структурирования в случае ООП связаны с различными аспектами базового понимания предметной задачи, которое требуется для оптимального управления соответствующей моделью:

- абстракция для выделения в моделируемом предмете важного для решения конкретной задачи по предмету, в конечном счёте – контекстное понимание предмета, формализуемое в виде класса;
- инкапсуляция для быстрой и безопасной организации собственно иерархической управляемости: чтобы было достаточно простой команды «что делать», без одновременного уточнения как именно делать, так как это уже другой уровень управления;
- наследование для быстрой и безопасной организации родственных понятий: чтобы было достаточно на каждом иерархическом шаге учитывать только изменения, не дублируя всё остальное, учтённое на предыдущих шагах;
- полиморфизм для определения точки, в которой единое управление лучше распараллелить или наоборот – собрать воедино.

1 ПОСТАНОВКА ЗАДАЧИ

Исследовать способы и методы обработки изображения. Реализовать приложение с графическим интерфейсом для обработки некоторых характеристик изображения и сохранения итогового изображения.

Программа для редактирования фотографий должна содержать классы для операций над изображением и классы, описывающий фильтры для изображения.

Для максимальной производительности и для того, чтобы избежать задержек в графическом интерфейсе, для обработки изображения следует реализовать отдельный поток. Благодаря этому пользователь сможет в реальном времени наблюдать изменения, которые он производит с изображением.

В качестве функций, которые можно применить к изображению, следует реализовать изменение яркости, контрастности, насыщенности, четкости и температуры, а также добавить возможность поворота изображения на 90 градусов. В процессе последующей разработки и обновления приложения можно будет добавлять и другие функции для работы с изображением.

Чтобы пользователь имел быстрый доступ к изображениям, с которыми он недавно работал, следует реализовать небольшой список недавно открытых изображений.

Также следует реализовать сохранение настроек, установленных пользователем, для дальнейшего их применения в редактирование других изображений. Для этого будет реализован пользовательский фильтр, в котором пользователь сможет сохранять текущие настройки изображения.

Кроме того, следует реализовать фильтры, которые в целом будут менять цветовую составляющую изображения. Например, инверсия цвета и перевод цвета в оттенки серого. А также следует предусмотреть фильтр, который будет возвращать изображение к первоначальному состоянию, чтобы пользователь мог отменить все свои изменения.

Следует предусмотреть перевод приложения на другие языки, чтобы большее количество людей могло им беспрепятственно пользоваться.

В приложении следует реализовать работу с изображениями форматов .png и .jpg, как с самыми распространёнными форматами изображений.

Также следует предусмотреть автоматическую подгонку объектов окна под его размер, чтобы пользователь мог менять размер окна так, как ему удобно.

2 ОБЗОР ЛИТЕРАТУРЫ

2.1 Обзор методов и алгоритмов решения поставленной задачи

Для обработки изображения был использован фреймворк OpenCV. Он позволяет представить изображение в виде матрицы пикселей. Благодаря этому, обращаясь к пикселям и меняя их значение, мы можем редактировать исходное изображение. Также OpenCV предоставляет ряд функций и методов, которые позволяют редактировать всё изображение сразу, например, изменить его яркость и контрастность, инвертировать изображение и другие возможности.

В ходе создания приложения было реализован контейнер двунаправленного кольца MyRing<T>. В нём реализованы методы для добавления и удаления элементов, методы смещения указателя на следующий или предыдущий элемент, а также перегружен оператор [] для получения доступа к произвольному элементу кольца, начиная счет от «головы» кольца. Данное кольцо используется для хранения фильтров и работы с ними.

Для реализации пользовательского интерфейса был использован фреймворк Qt. Программа Qt Creator позволяет легко и быстро создать сам интерфейс приложения [4], как показано на рисунке 2.1, а благодаря системе сигналов и слотов, которая представленная данным фреймворком, данный интерфейс соединяется с программным кодом. Также, благодаря реализованному в Qt классу потока QThread, был реализован производный от него класс MyThread, в котором происходит обработка изображения.

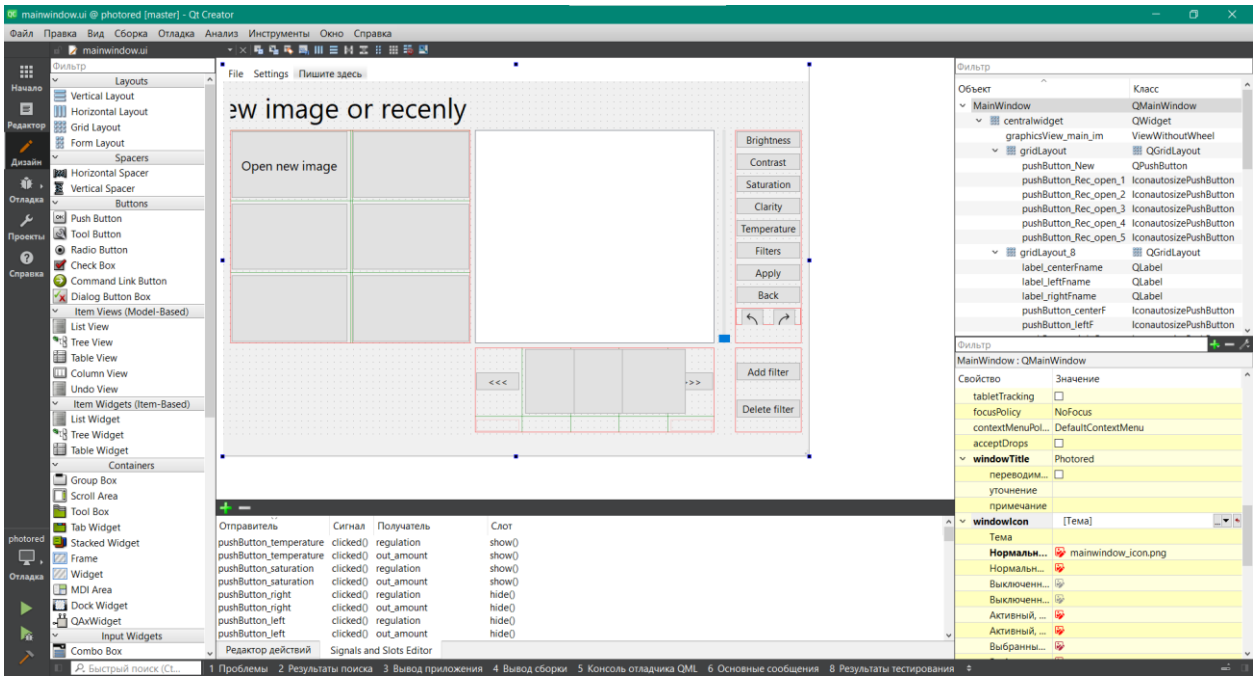

Рисунок 2.1 – Qt Creator

2.2 Обзор аналогов приложения

2.2.1 Adobe Photoshop Lightroom

Adobe Photoshop Lightroom – графический редактор компании Adobe для работы с цифровыми фотографиями [5]. Может использоваться для «проявки» «цифровых негативов» (форматы данных DNG, Raw), ретуши фотоснимков и организации их каталога. Особенностью программы является «недеструктивное редактирование», при котором исходный файл изображения остаётся неизменным, а все операции редактирования изображения осуществляются над автоматически сгенерированными из мастер-файла рабочими файлами – «версиями». На рисунке 2.2 представлено данное приложение.

2.2.2 Microsoft Photos

Microsoft Photos (Фотографии) – программа для просмотра изображений, видео-редактор, сортировщик фотографий, редактор растровой графики и приложение для раздачи фотографий [6]. «Фотографии» предоставляет базовые возможности растрового графического редактора, такие как: коррекция экспозиции или цвета, изменение размера, обрезка, удаление «эффекта красных глаз», удаление «пятен», удаление «шумов». Приложение показано рисунке 2.3.

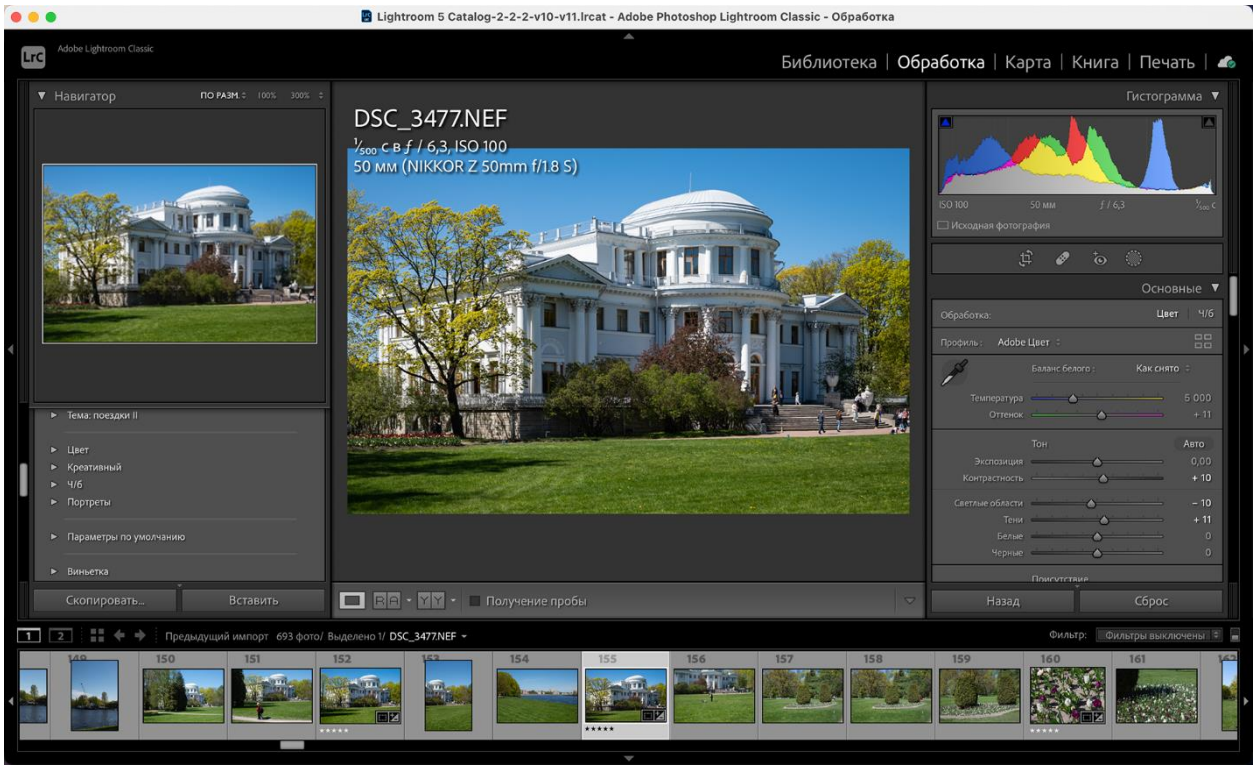

Рисунок 2.2 – Adobe Photoshop Lightroom

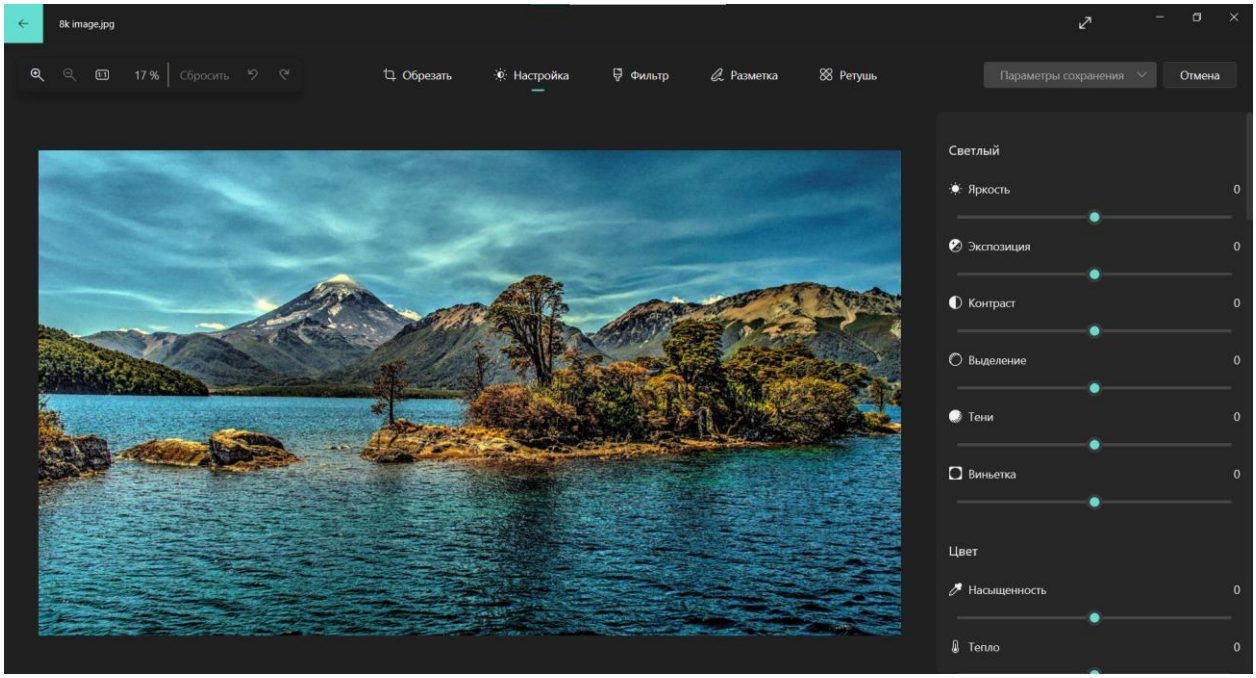

Рисунок 2.3 – Microsoft Photos (Фотографии)

2.2.3 GNU Image Manipulation Program

GNU Image Manipulation Program или GIMP – свободно распространяемый растровый графический редактор, программа для создания и обработки растровой графики и частичной поддержкой работы с векторной графикой [7]. В GIMP присутствует набор инструментов цветокоррекции: кривые, уровни, микшер каналов, постеризация, тон-насыщенность, баланс цветов, яркость-контраст, обесцвечивание. Дополнительные возможности по коррекции изображений на протяжении всей работы реализованы в виде экранных фильтров. К ним относятся: имитация разных типов дальтонизма (протанопия, дейтеранопия, тританопия), гамма-коррекция, коррекция контраста, управление цветом. Интерфейс изображения представлен на рисунке 2.4.

2.2.4 PhotoFiltre

PhotoFiltre – растровый графический редактор для операционной системы Windows [8]. В PhotoFiltre позволяет регулировать яркость, контрастность, насыщенность, исправлять гамму, пользоваться всевозможными фильтрами (акварель, пастель, чернила и т. д.), работать со слоями. Кроме того, в этом редакторе можно работать с декоративным текстом, оптимизировать графику, создавать поздравительные конверты и открытки из готовых шаблонов. Данное приложение изображено на рисунке 2.5.

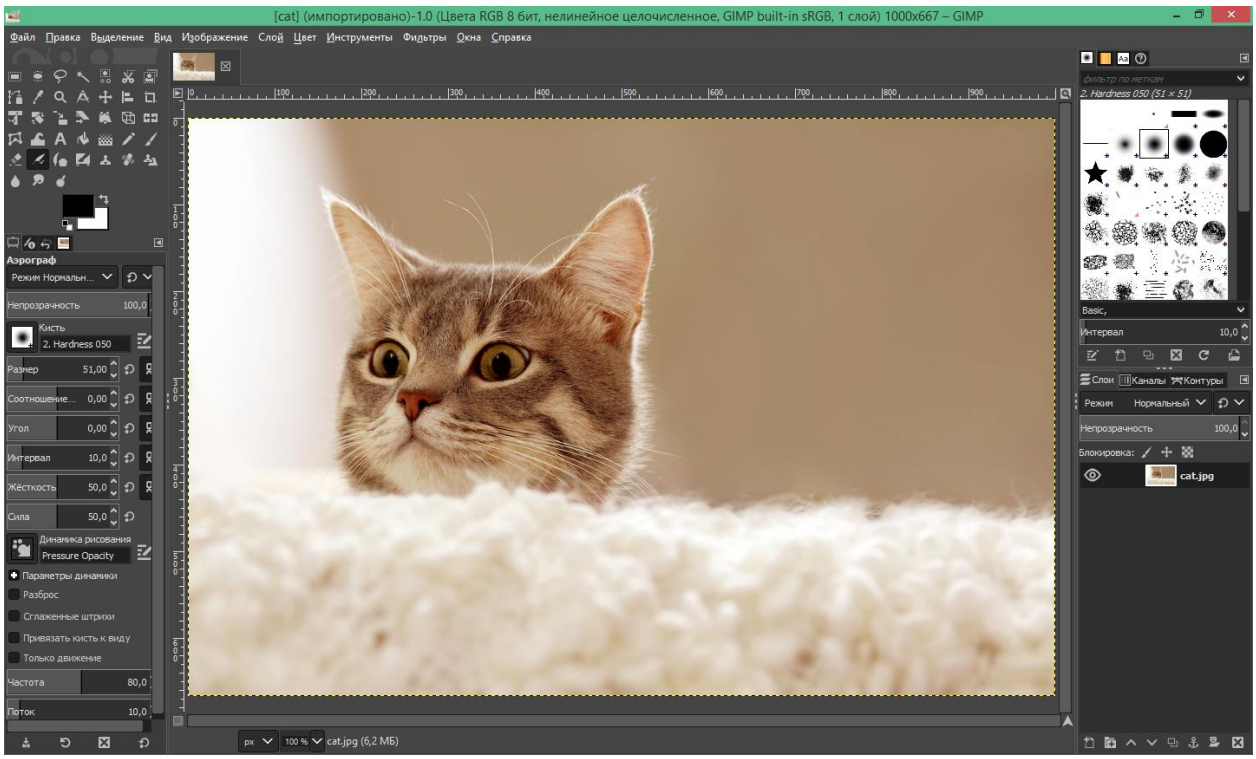

Рисунок 2.4 – GNU Image Manipulation Program

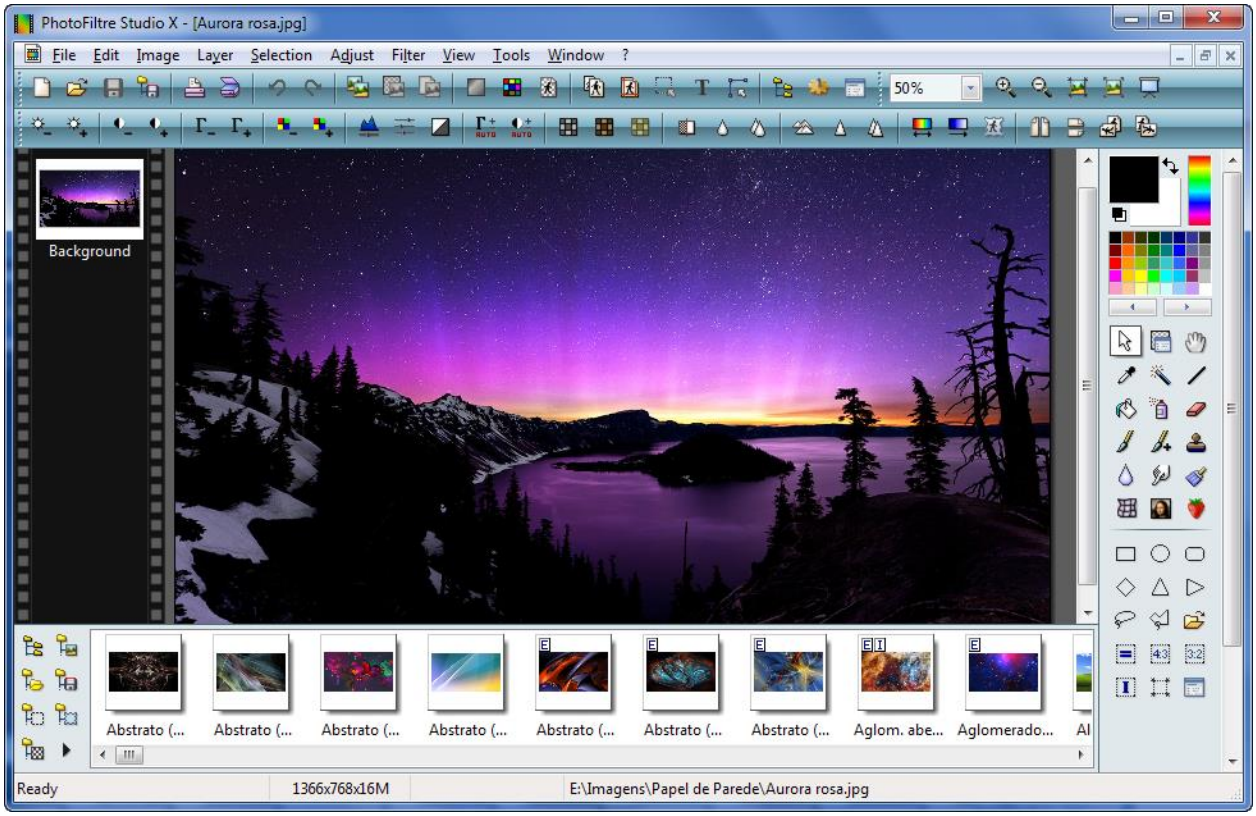

Рисунок 2.5 – PhotoFiltre

2.2.5 Adobe Photoshop

Adobe Photoshop – многофункциональный растровый графический редактор, разрабатываемый и распространяемый компанией Adobe Systems [9]. В основном работает с растровыми изображениями, однако имеет некоторые векторные инструменты. Продукт является лидером рынка в области коммерческих средств редактирования растровых изображений и наиболее известной программой разработчика. Поддерживается обработка изображений с глубиной цвета 8 бит (256 градаций на один канал), 16 бит (используется 15 бит плюс один уровень, то есть 32 769 уровней) и 32 бита (используются числа одинарной точности с плавающей запятой). Возможно сохранение в файле дополнительных элементов, как: направляющих (Guide), каналов (например канала прозрачности — Alpha channel), путей обтравки (Clipping path), слоёв, содержащих векторные и текстовые объекты. Файл может включать цветовые профили (ICC), функции преобразования цвета (transfer functions). Допускаются неквадратные пиксели (Pixel Aspect Ratio). На рисунке 2.6 представлено рабочее окно приложения.

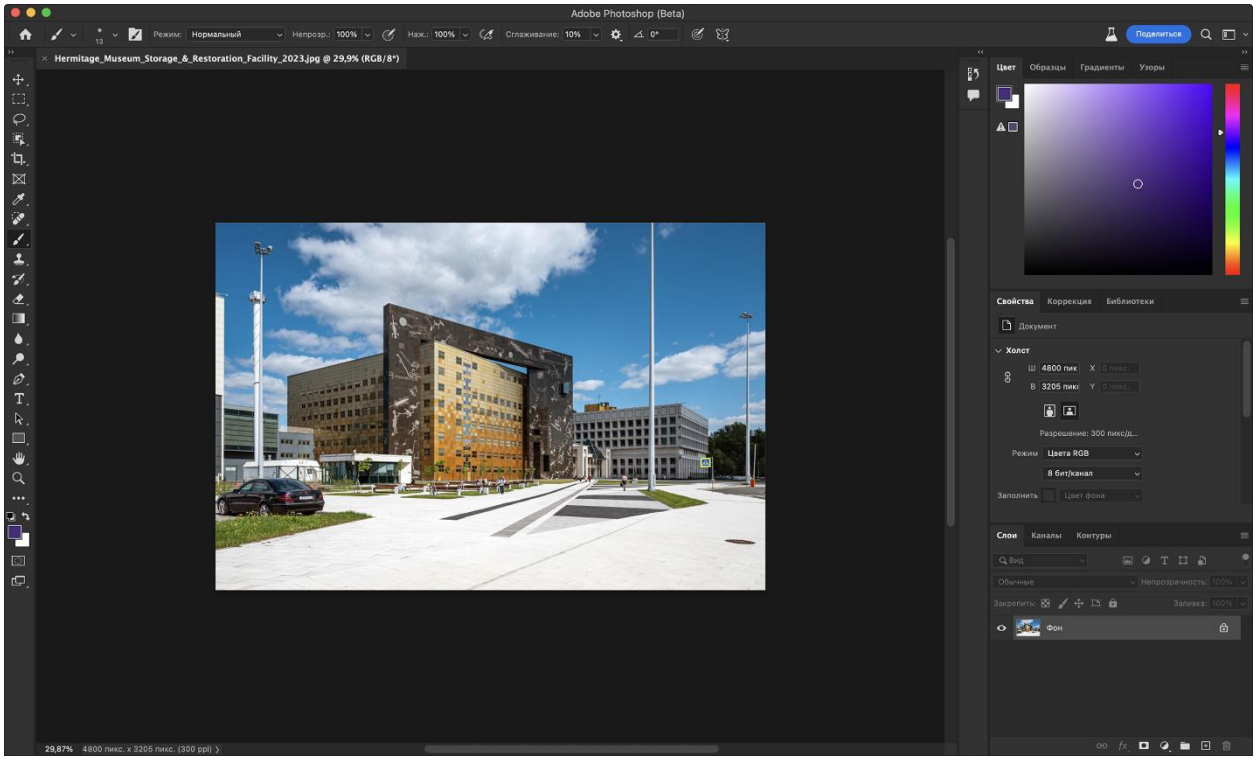

Рисунок 2.6 – Adobe Photoshop

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описываются входные и выходные данные программы, диаграмма классов, а также приводится описание используемых классов и их методов.

3.1 Структура входных и выходных данных

Таблица 3.1 – файл с пользовательскими фильтрами filters_inform.json

Название	Яркость	Контрастность	Насыщенность	Четкость	Температура
Filter1	13	30	36	0	-26

Таблица 3.2 – файл ранее открытых изображений recently_opened.json

Путь к файлу
D:\\University\\cs\\sem3\\cursach\\test.png

3.2 Разработка диаграммы классов

Диаграмма классов для данного курсового проекта представлена в приложении А.

3.3 Описание классов

3.3.1 Классы операций над изображением

Класс `Operation` является абстрактным классом. Он описывает операцию над изображением.

Поля класса:

- `int value` – значение для изменения характеристики изображения;
- `cv::Mat image` – изображения в виде матрицы, предоставляемое OpenCV.

Методы класса:

- `virtual cv::Mat exec() = 0` – чисто виртуальная функция, которая после переопределения в производных класса будет производить обработку изображения `image` и возвращать обработанное изображение;

От класса `Operation` наследуются классы, каждый из которых будет обрабатывать свою конкретную характеристику изображения.

Класс `Oper_brightness` предназначен для изменения яркости изображения. Является производным от класса `Operation`.

Поля класса наследуются от класса `Operation`.

Методы класса:

- `Oper_brightness(int, cv::Mat)` – конструктор класса, который устанавливает значение характеристики и изображение для обработки;
- `virtual cv::Mat exec() override` – виртуальная функция, которая является переопределением метода из базового класса. Она изменяется яркость изображения и возвращает измененное изображение.

Класс `Oper_contrast` предназначен для изменения контрастности изображения. Является производным от класса `Operation`.

Поля класса наследуются от класса `Operation`.

Методы класса:

- `Oper_contrast(int, cv::Mat)` – конструктор класса, который устанавливает значение характеристики и изображение для обработки;
- `virtual cv::Mat exec() override` – виртуальная функция, которая является переопределением метода из базового класса. Она изменяется контрастность изображения и возвращает измененное изображение.

Класс `Oper_saturation` предназначен для изменения насыщенности изображения. Является производным от класса `Operation`.

Поля класса наследуются от класса `Operation`.

Методы класса:

- `Oper_saturation(int, cv::Mat)` – конструктор класса, который устанавливает значение характеристики и изображение для обработки;
- `virtual cv::Mat exec() override` – виртуальная функция, которая является переопределением метода из базового класса. Она изменяется насыщенность изображения и возвращает измененное изображение.

Класс `Oper_clarity` предназначен для изменения четкости изображения. Является производным от класса `Operation`.

Поля класса наследуются от класса `Operation`.

Методы класса:

- `Oper_clarity(int, cv::Mat)` – конструктор класса, который устанавливает значение характеристики и изображение для обработки;
- `virtual cv::Mat exec() override` – виртуальная функция, которая является переопределением метода из базового класса. Она изменяется четкость изображения и возвращает измененное изображение.

Класс `Oper_temperature` предназначен для изменения температуры изображения. Является производным от класса `Operation`.

Поля класса наследуются от класса `Operation`.

Методы класса:

- `Oper_temperature(int, cv::Mat)` – конструктор класса, который устанавливает значение характеристики и изображение для обработки;
- `virtual cv::Mat exec() override` – виртуальная функция, которая является переопределением метода из базового класса. Она изменяется температуру изображения и возвращает измененное изображение.

3.3.2 Класс потока обработки изображения

Класс потока `MyThread` предназначен для обработки изображения параллельно с основным потоком программы.

Поля класса:

- `std::queue<Operation *> queue` – очередь указателей на базовый класс `Operation`, которая хранит порядок выполнения операций над изображения.

Методы класса:

- `void push(Operation *)` – метод для помещения операции над изображением в очередь;
- `virtual void run() override` – переопределение виртуальной функции, вызывается при запуске потока, выполняет операцию из вершины очереди и отправляет это изображение вместе с сигналом `signalGUI(cv::Mat)`.

Сигналы класса:

- `signalGUI(cv::Mat)` – сигнал, который сообщает о выполнении операции над изображение и передает обработанное изображение.
- `void terminateThread()` – функция, которая завершает поток.

Благодаря классу `MyThread` пользователь может в реальном времени видеть изменения изображение, при изменении положения ползунка характеристики.

3.3.3 Классы фильтров изображения

Класс `Filter` является абстрактным классом. На его основе с помощью наследования реализованы классы `Inverse`, `Original`, `Gray` и `CustomFilter`.

Поля класса:

- `cv::Mat image` – обработанное изображение;
- `std::string name` – строка для хранения названия фильтра;
- `int brightness` – значение яркости изображение;
- `int contrast` – значение контрастности изображения;
- `int saturation` – значение насыщенности изображения;
- `int clarity` – значение четкости изображения;

- `int temperature` – значение температуры изображения.

Методы класса:

- `cv::Mat apply()` – метод для возвращения обработанного изображения;
- `std::string get_filter_name()` – метод для получения названия фильтра;
- `int get_brightness()` – метод для получения значения яркости фильтра;
- `int get_contrast()` – метод для получения значения контрастности фильтра;
- `int get_saturation()` – метод для получения значения насыщенности фильтра;
- `int get_clarity()` – метод для получения значения четкости фильтра;
- `int get_temperature()` – метод для получения значения температуры фильтра;
- `virtual ~Filter() = 0` – чисто виртуальный деструктор, который делает класс абстрактным.

Класс `Inverse`, который предназначен для инверсии изображения.

Поля класса наследуются от базового класса `Filter`.

Методы:

- `Inverse(cv::Mat)` – конструктор, которому передается изображение для обработки. Он инвертирует это изображение и сохраняет итоговый результат, задается название фильтра.

Класс `Original`, который предназначен для получения первоначального изображения.

Поля класса наследуются от базового класса `Filter`.

Методы:

- `Original(cv::Mat)` – конструктор, которому передается изображение для обработки, сохраняет это изображение, задает название фильтра.

Класс `Gray`, который меняет цветовую гамму изображения на серую.

Поля класса наследуются от базового класса `Filter`.

Методы:

- `Gray(cv::Mat)` – конструктор, которому передается изображение для обработки. Он конвертирует это изображение в оттенках серого и сохраняет итоговый результат, задается название фильтра.

Класс `CustomFilter`, который предназначен для применения к изображению характеристик, заданных пользователем.

Поля класса наследуются от базового класса `Filter`.

Методы:

- `CustomFilter(std::string, cv::Mat, int, int, int, int, int)` – конструктор, которому передается название фильтра, изображение

для обработки, значения яркости, контрастности, насыщенности, четкости и температуры. Он обрабатывает изображение по заданным параметрам и сохраняет итоговый результат, также сохраняется название фильтра и переданные значения характеристик.

3.3.4 Класс кнопки с автоматическим изменением размера иконки

Класс `IconautosizePushButton` предназначен для автоматического изменения размера иконки кнопки во время изменения размера самой кнопки. Является производным классом от класса `QPushButton`

Поля класса:

Поля, наследуемые от класса `QPushButton`.

- `QString image_path` – строка, в которой хранится путь до изображения иконки.

Методы класса:

- `void set_image_path(QString &)` – задает строку, в которой хранится путь до изображения иконки;
- `QString &get_image_path()` – возвращает строку, в которой хранится путь до изображения иконки.
- `void resizeEvent(QResizeEvent *) override` – переопределение функции изменения размера кнопки, которая меняет и размер кнопки, и размер иконки.

3.3.5 Классы оконных интерфейсов

Класс `MainWindow` является основным оконным интерфейсов, в котором происходит открытие, обработка и экспорт изображения. Наследуется от класса `QMainWindow` и класса `Ui_MainWindow`, который сгенерирован автоматические и в котором объявлены все объекты, которые помещены на окно с помощью Qt Creator.

Поля класса:

- Поля, наследуемые от базовых классов;
- `MyThread *mythread` – поток для обработки изображения;
- `PROCESSES current_process` – хранит текущий процесс над изображением;
- `MyRing<Filter *>filters` – двунаправленное кольцо, которое хранит существующие фильтры;
- `int filter_number` – хранит номер выбранного фильтра;
- `QTranslator qtlangtransl` – перевод приложения на другие языки;
- `QGraphicsScene *graphicsScene` – графическая сцена для отображения графических предметов на графическом виде;
- `QGraphicsPixmapItem *pixmap` – графический предмет для отображения изображения типа QPixmap на графической сцене;

- `image_info image_info` – информация об изображении.

Методы класса:

- `void set_connections()` – производит основные соединения сигналов со слотами;
- `MainWindow(QWidget *)` – конструктор главного окна, в котором инициализируются переменные, выделяется память под указатели, скрываются ненужные в начальный момент объекты окна и запускается поток обработки изображения;
- `void set_curr_proc(PROCESSES)` – задает, какой процесс сейчас происходит;
- `PROCESSES get_curr_proc()` – возвращает, какой процесс сейчас происходит;
- `void prepare_image()` – подготавливает стартовое изображение к дальнейшим операциям;
- `void set_filters()` – инициализирует кольцо фильтров, считывая значения из файла;
- `void save_filters()` – сохраняет пользовательские фильтры в файл;
- `cv::Mat get_filtered_image(int)` – получает номер фильтра, возвращает изображение с примененным к нему фильтром;
- `std::string get_filter_name(int)` – возвращает имя фильтра, номер которого передан;
- `void set_filter_number(int)` – задает значение переменной `filter_number`;
- `void next_prev_filter(int)` – перемещает «голову» кольца на следующий элемент, если передаваемое число положительное, или на предыдущий элемент, если передаваемое число отрицательное;
- `void resizeEvent(QEvent *) override` – переопределение обработчика событий, которое, если зафиксировано событие изменения языка приложение, запускает изменение переводимых надписей на объектах окна;
- `void change_image(cv::Mat)` – изменяет отображаемое изображение, на передаваемое;
- `void set_rec_opened_butts()` – задает кнопки ранее открытых изображений;
- `void start_proc(QString &)` – открывает изображение по переданному пути, либо, если строка пустая, открывает файловое диалоговое окно, где пользователь выбирает изображение для обработки. Скрывает объекты для открытия изображения и отображает объекты для работы с изображением. Сохраняет путь до открытого изображения, если этот не был сохранен ранее;
- `void main_proc(int)` – основной процесс работы с изображением. Получает значение с ползунка и меняет определенную характеристику изображение в зависимости от текущего процесса;

- `void set_slider_limits()` – задает границы ползунка, а также его начальное значение;
- `void end_main_proc()` – сохраняет значение характеристики изображение, с которой только что работали;
- `void rotate_left()` – поворачивает изображение на 90 градусов против часовой стрелки;
- `void rotate_right()` – поворачивает изображение на 90 градусов по часовой стрелки;
- `void save_image()` – сохраняет изображение в выбранное пользователем место;
- `void set_new_image()` – скрывает объекты для работы с изображением и отображает объекты для открытия изображения, обнуляет процесс и характеристики;
- `void set_filters_buttons()` – задает иконку кнопки и отображение названия фильтра в зависимости от расположения фильтров в кольце;
- `void set_deleteF_enabled(std::string)` – задает активность кнопки в зависимости от имени выбранного фильтра;
- `void back_from_filters()` – возвращает пользователя от выбора фильтра, к изменению характеристик изображения;
- `void apply_filter()` – применяет фильтр к изображению;
- `void delete_filter()` – удаляет выбранный фильтр;
- `void add_filter()` – добавляет фильтр в коллекцию, значения фильтра берутся из значений изображения, настроенных пользователем на данный момент;
- `void show_pressed_button()` – визуально помечает, какая кнопка сейчас нажата;
- `void change_language(const char*)` – изменяет язык приложения.

Класс `FilterName_window`, с помощью которого задается имя для добавляемого фильтра. Он наследуется от класса `QWidget` и класса `Ui_Form`, который сгенерирован автоматически на основании созданного в Qt Creator окна.

Поля класса:

- Поля, наследуемые от базовых классов;
- `MyRing<Filters*> *filters` – указатель на двунаправленное кольцо фильтров;
- `std_string filter_name` – имя фильтра, введенное пользователем.

Методы класса:

- `FilterName_window(QWidget *)` – конструктор класса, в котором создаются объекты окна, а также происходят соединения сигналов со слотами;
- `void set_filters(MyRing<Filter*>*)` – инициализирует указатель на кольцо фильров;

- `bool is_name_in_filters(std::string)` – ищет указанное имя среди всех фильтров;
- `std::string get_filter_name()` – возвращает `filter_name`;
- `void changeEvent(QEvent *) override` – переопределение обработчика событий, который ри сменен языка приложение запускает изменение надписей объектов окна;
- `void safe_filter_name()` – проверяет имя, вводимое пользователем, и, если имя не пустая строка и такого имени нет среди всех фильтров, то сохраняет его в `filter_name`.

Сигналы класса:

- `void filter_name_got()` – сообщает о том, что имя фильтра получено.

3.3.6 Класс двунаправленного кольца

Класс RingNode<T>, который является отдельным звеном кольца.

Поля класса:

- `T data` – информация, что хранится в звене;
- `RingNode *next` – указатель на следующее звено;
- `RingNode *prev` – указатель на предыдущее звено.

Методы класса:

- `RingNode()` – конструктор по умолчанию;
- `RingNode(T)` – конструктор, инициализирующий значение data.

Класс MyRing<T> –двунаправленное кольцо, построенное из звеньев RingNode<T>.

Поля класса:

- `RingNode<T> *head` – указатель на «голову» кольца;
- `int ring_size` – размер кольца.

Методы класса:

- `MyRing()` – конструктор по умолчанию;
- `MyRing(const MyRing &)` – конструктор копирования;
- `~MyRing()` – деструктор класса;
- `void push(T)` – добавление элемента в кольцо;
- `void pop_head()` – удаление звена, на которое указывает «голова»;
- `T get_data()` – получение значения из «головы» кольца;
- `void next_node()` – перемещение «головы» на следующее звено;
- `void prev_node()` – перемещение «головы» на предыдущее звено;
- `bool empty()` – проверка, пустое ли кольцо;
- `int size()` – получение размера кольца;
- `void clean()` – очистка кольца;
- `T &operator[] (const int)` – перегрузка оператора `[]`, которая возвращает значение из указанного звена.

3.3.7 Другие классы

Класс графического вида `ViewWithoutWheel`, который игнорирует события колёсика мыши и подгоняет размер изображения под размер окна. Наследуется от класса `QGraphicsView`.

Методы класса:

- `virtual void wheelEvent(QWheelEvent *) override` – переопределение обработчика событий колёсика мыши, которые игнорирует колёсико мыши;
- `virtual void resizeEvent(QResizeEvent *) override` – переопределение обработчика изменения размера, который подгоняет размер изображения под размер окна.

Перечисление `PROCESSES`, которое содержит процессы, которые могут происходить с изображением.

Константы перечисления:

- `BRIGHTNESS` – изменение яркости;
- `CONTRAST` – изменение контрастности;
- `SATURATUIN` – изменение насыщенности;
- `CLARITY` – изменение четкости;
- `TEMPERATURE` – изменение температуры;
- `ROTATION` – поворот изображения;
- `FILTER` – применение фильтров;
- `NON` – изменение изображения не происходит.

Перечисление `FILTER`, которое содержит все возможные типы фильтров.

Константы перечисления:

- `INVERSE` – инверсия изображения;
- `ORIGINAL` – оригинальное изображение;
- `GRAY` – изображение в оттенках серого;
- `CUSTOM` – пользовательский фильтр.

Структура image_info, которая хранит начальное изображение и измененные данные.

Поля структуры:

- `QPixmap *start_image` – начальное изображение;
- `QPixmap *image_in_proc` – обработанное изображение;
- `int brightness` – значение яркости обработанного изображения;
- `int contrast` – значение контрастности обработанного изображения;
- `int saturation` – значение насыщенности обработанного изображения;
- `int clarity` – значение четкости обработанного изображения;
- `int temperature` – значение температуры обработанного изображения.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Разработка алгоритмов

`void MainWindow::start_proc(QString &)` — функция, которая предназначена для открытия изображения и задания начальных параметров обработки.

Алгоритм по шагам:

1. Начало
2. Если путь до изображения пуст, то запустить файловый диалог. Иначе перейти к шагу 4.
3. Если путь пуст, то перейти к шагу 12.
4. Прочитать изображение по имеющемуся пути.
5. Сохранить прочитанное изображение в переменную, хранящую первоначальное изображение.
6. Вывести прочитанное изображение.
7. Открыть файл недавно открытых изображений, в котором хранятся пути до недавно открытых изображений.
8. Если текущий путь не присутствует в файле, то перейти к шагу 9, иначе перейти к шагу 11.
9. Если количество хранящихся путей меньше пяти, то добавить текущий путь в файле и перейти к шагу 11, иначе перейти к пункту 10.
10. Удалить первый путь из файла и занести в него новый путь.
11. Закрыть файл.
12. Конец.

`void FilterName_window::save_filter_name()` — функция для проверки введенного пользователем имени нового фильтра и, если оно проходит проверку, сохранения этого имени.

Алгоритм по шагам:

1. Начало.
2. Получение имени, введенного пользователем.
3. Пока первый символ имени равен символу пробела, удалять первый символ.
4. Пока последний символ имени равен символу пробела, удалять последний символ.
5. Если имя пусто, вывести сообщение о некорректности имени и перейти к шагу 9.
6. Если введенное имя совпадает с уже существующем имени, вывести сообщение о том, что данное имя уже занято, и перейти к шагу 9.
7. Сохранить введенное имя.
8. Выдать сигнал о том, что имя нового фильтра введено корректно.
9. Конец.

4.2 Разработка схем алгоритмов

Схема алгоритма метода `void start_proc(QString &)` класса `MainWindow` представлена в приложении Б. Данный метод вызывает каждый раз, как пользователь выбирает изображение для работы. Если пользователь выбрал открыть новое изображение, то срока, передаваемая в метод, будет пустой, вследствие чего будет вызван файловый диалог, где пользователь выбирает нужное ему изображение из памяти компьютера. Если же пользователь выбрал открыть ранее открытое изображение, то в метод будет передан путь до выбранного изображения в памяти изображения. После открытия изображения оно сохраняется в приложение для дальнейшей работы, а путь до открытого изображения заносится в список ранее открытых изображений, если он там не присутствует.

Схема алгоритма метода `void save_filter_name()` класса `FilterName_window` представлена в приложении В. Данная функция предназначена для проверки введенного пользователем имени нового фильтра. Для начала из имени удаляются все пробелы в начале и конце строки. Далее имя проверяется по следующим критериям: пустое ли имя и существует ли уже такое имя среди всех фильтров программы. Если введенное имя не пустое и является уникальным среди всех фильтров, то это имя сохраняется и основной программе сообщается об удачном введении имени для запуска процесса добавления нового фильтра к списку имеющихся.

При разработке функций часто бывает полезно сначала продумать общий алгоритм действий, составить схему алгоритма, а потом реализация становится очень простой.

При разработке алгоритма по шагам и схемы алгоритма важно не включать детали реализации, которые никак не помогают понять суть алгоритма. Например, какие-то специфические особенности разных операционных систем, языков программирования и не только вряд ли будут указаны на схеме алгоритма. Схему алгоритма можно описать простыми словами, иногда с указанием сторонних функций.

В данном случае не была включена обработка путей к файлам, так как в разных операционных системах пути описываются по-разному, а также опущены моменты работы с объектами окна интерфейса.

5 РЕЗУЛЬТАТЫ РАБОТЫ

При запуске программы пользователя приветствует окно с выбором: открыть новое изображение для обработки или одно из пяти ранее открытых, как показано на рисунке 5.1.

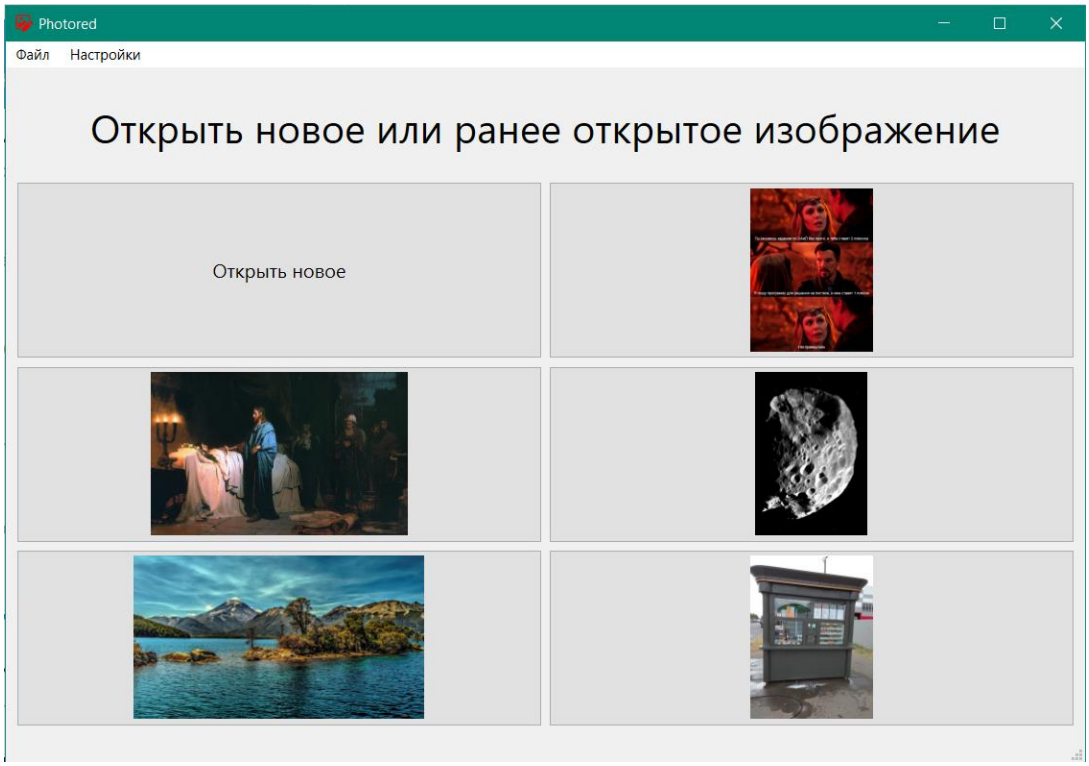

Рисунок 5.1 – Начальное окно

Если пользователь выбирает открыть новое изображение, то открывается диалоговое окно, показанное на рисунке 5.2, где он должен выбрать изображение с расширением .png или .jpg, которое хранится на компьютере.

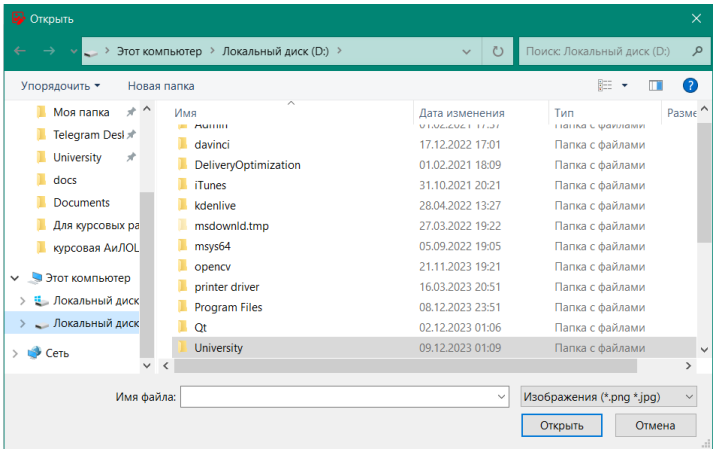

Рисунок 5.2 – Диалоговое окно для открытия нового изображения

После выбора изображения оно отображается на большей части окна, а в правой части появляются кнопки для редактирования характеристик изображения (яркость, контрастность, насыщенность, резкость, температура), применения фильтров и поворота изображения на 90 градусов. Рабочее окно показано на рисунке 5.3.

При выборе какой-нибудь характеристики для редактирования на экране появляется ползунок для изменения значения этой характеристики и отображение его текущего значения. Во время движения ползунка изображение меняется в реальном времени. На рисунке 5.4 показано изображение, с измененной контрастность в качестве примера.

Если пользователь желает повернуть свое изображение, то ему следует выбрать одну из кнопок (в зависимости от того, в какую сторону надо повернуть) на которой изображена изогнутая стрелка. Пример поворота показан на рисунке 5.5.

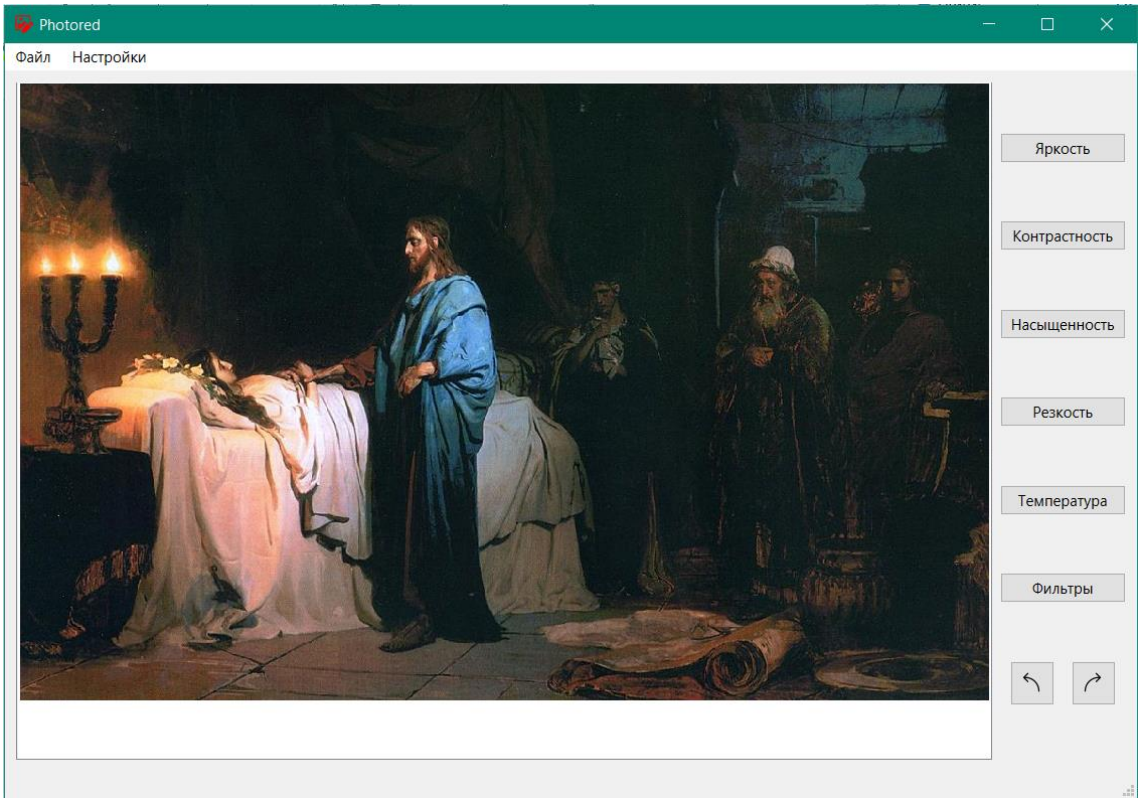

Рисунок 5.3 – Выбор операции для работы над изображением

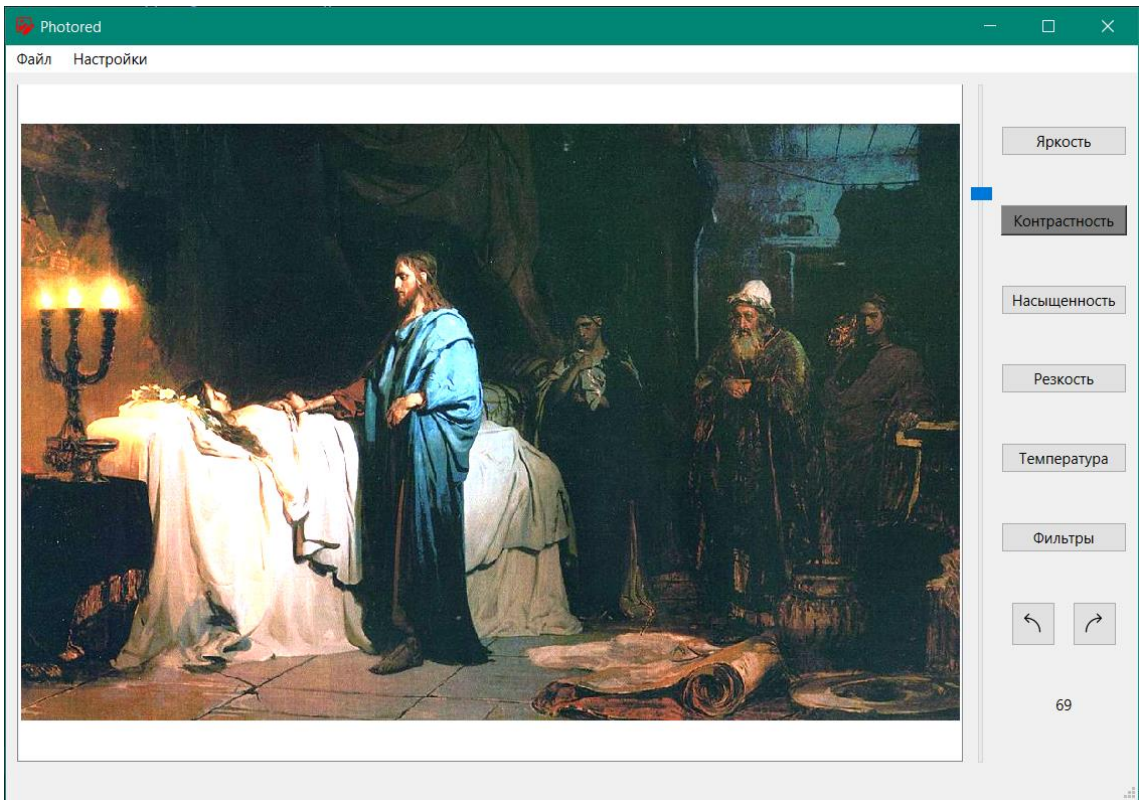

Рисунок 5.4 – Изменение характеристики изображения

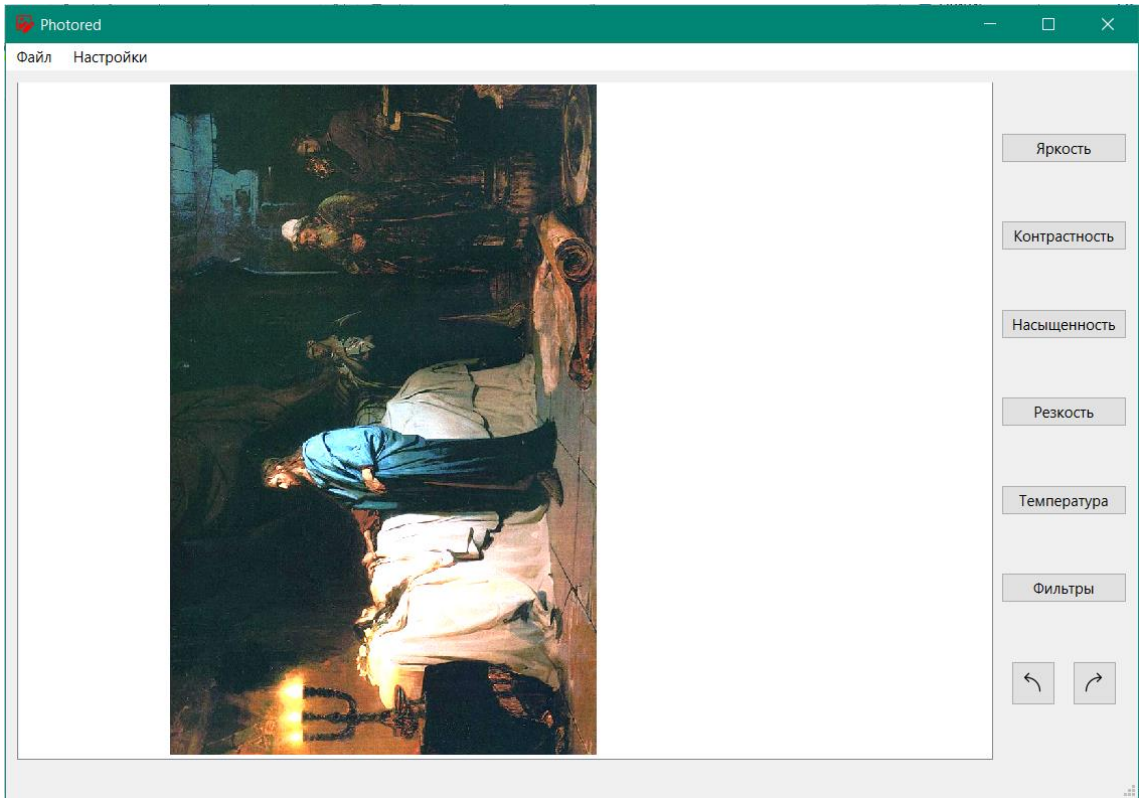

Рисунок 5.5 – Поворот изображения

Также в программе предусмотрено применение к изображению фильтров. Реализовано четыре типа фильтров: инверсия изображения, оригинальное изображение, изображение в оттенках серого и пользовательский фильтр, который применяет к изображению настроенные пользователем характеристики. При открытии фильтров кнопки редактирования характеристик исчезают, а под изображением появляется список существующих фильтров, который можно листать с помощью кнопок. При нажатии на фильтр основное изображение будет меняться под этот фильтр, чтобы пользователь мог лучше увидеть отфильтрованное изображение. На рисунке 5.6 показан пример выбора одного из фильтров.

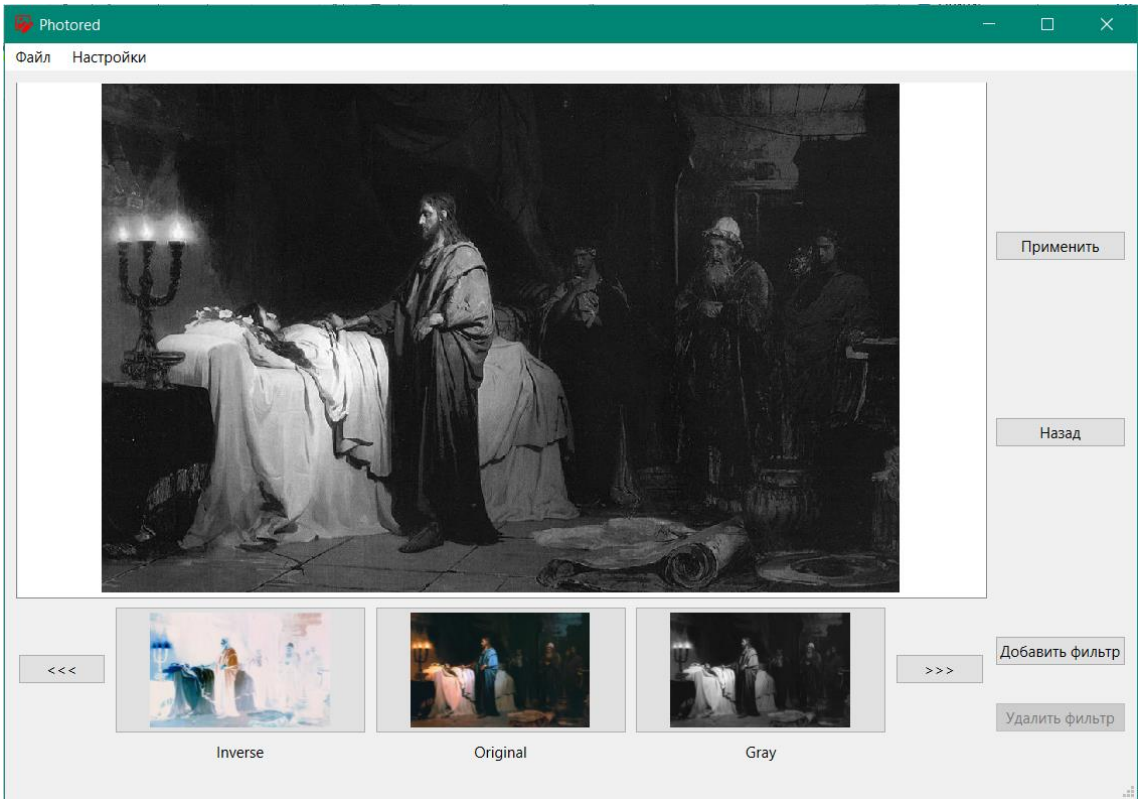

Рисунок 5.6 – Фильтры изображения

В разделе фильтров пользователь также имеет возможность добавления и удаления пользовательских фильтров. Чтобы добавить фильтр, пользователь должен заранее настроить нужные ему характеристики, после чего в открытых фильтрах он должен нажать кнопку «Добавить фильтр». После нажатия появляется окно, как на рисунке 5.7, где нужно ввести имя добавляемого фильтра. После сохранения фильтр будет доступен для использования в этом и последующих сеансах работы приложения, как показано на рисунке 5.8.

Для удаления фильтра, пользователю нужно выбрать пользовательский фильтр, после чего нажать кнопку «Удалить фильтр». После удаления данные о фильтре исчезают из данных программы.

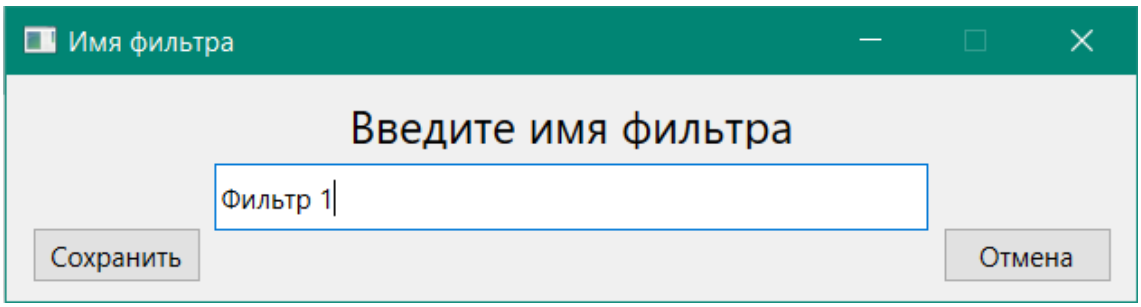

Рисунок 5.7 – Введение нового имени фильтра

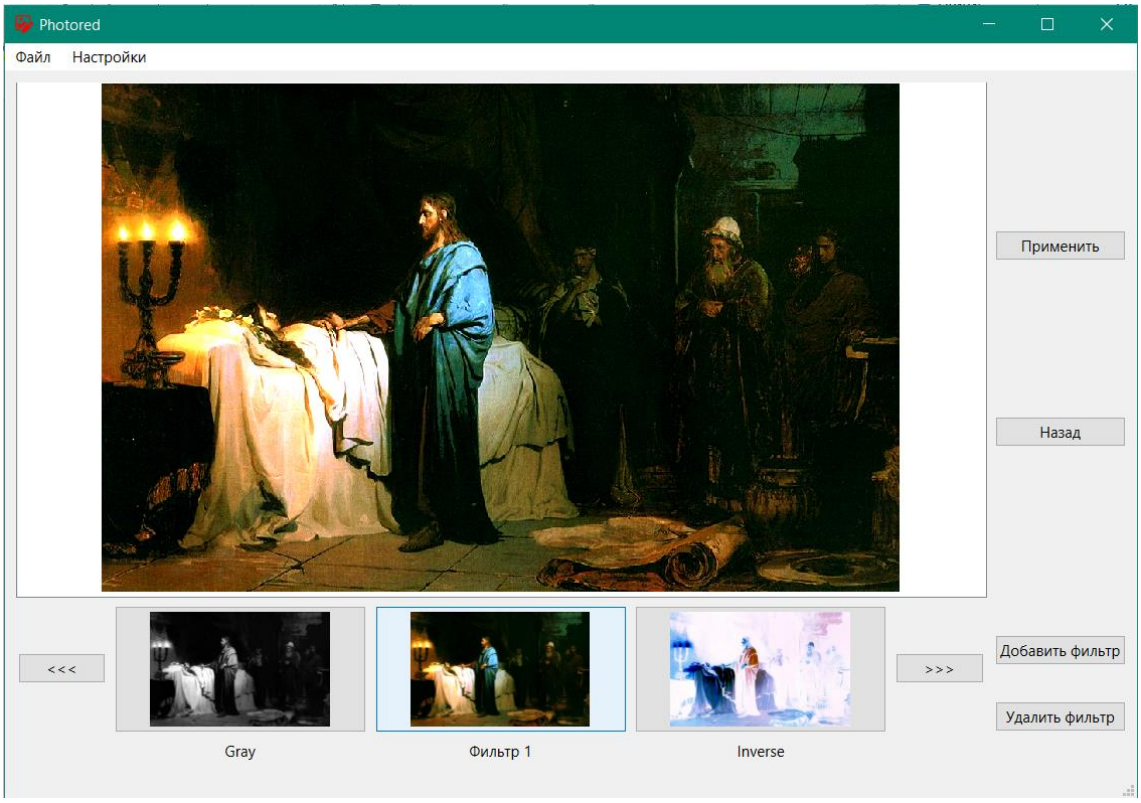

Рисунок 5.8 – Пользовательский фильтр в общем списке фильтров

После того, как пользователь закончил работу с изображением, он может сохранить итоговое изображение либо открыть новое изображение для работы. Для этого следует открыть подменю «Файл», как показано на рисунке 5.9.

При выборе «Новое изображение» окно приложения изменится на начальное. Если выбрать «Экспорт», то появится диалоговое окно, как на рисунке 5.10, где пользователь может выбрать место сохранения изображения, его название и тип (.png, .jpg).

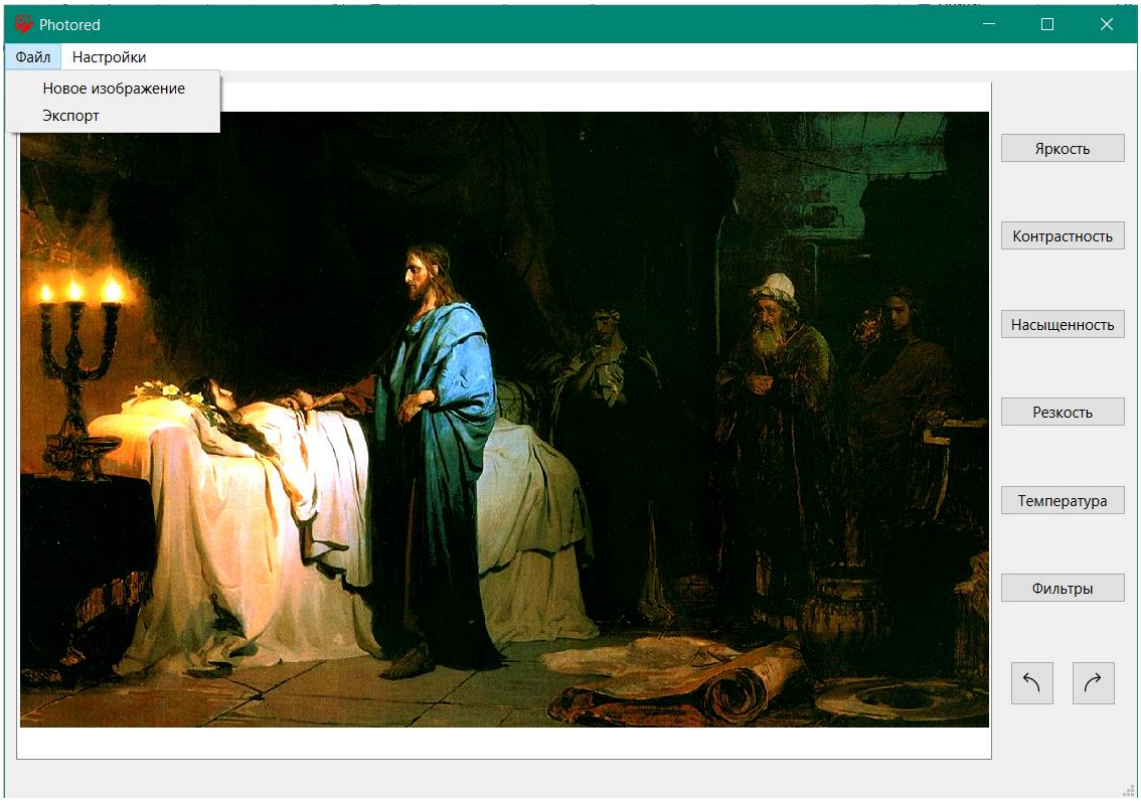

Рисунок 5.9 – Подменю «Файл»

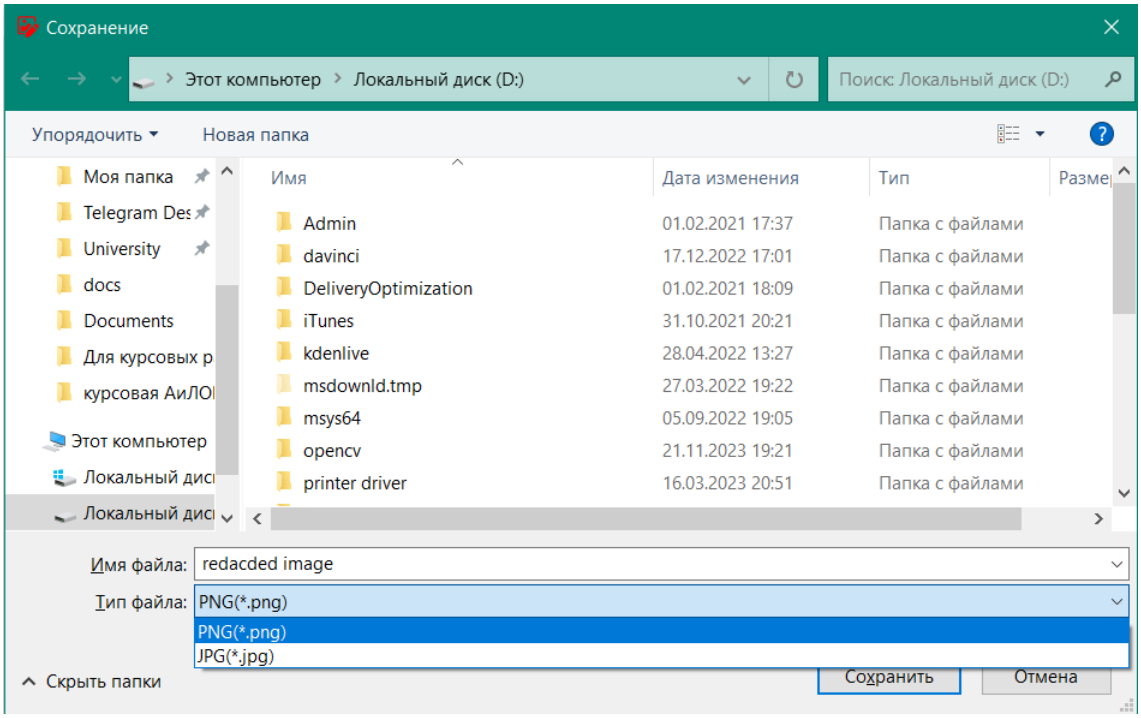

Рисунок 5.10 – Сохранение изображения

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсовой работы было создано приложение для редактирования изображений. Графический интерфейс приложения был создан с помощью фреймворка Qt, а обработка изображения основана на фреймворке OpenCV.

Было изучено представление изображения в памяти компьютера, разделения изображения на цветовые каналы, цветовые модели изображения. Также были изучены методы преобразования изображения, например, гамма-коррекция для изменения температуры изображения и ядро свёрстки для изменения четкости изображения. OpenCV оказался очень удобным и простым фреймворком для работы с изображением, так как он имеет свою переменную-матрицу cv::Mat, благодаря которой можно работать как с самим изображением, так и его отдельными компонентами (каналами, пикселями), а также в данном фреймворке представлено большое количество функций для всевозможной обработки изображения: разбиение и объединение каналов, изменение цветовой модели, применение функции ко всем пикселям и так далее.

Также было изучено разделение приложения на потоки для оптимизации приложения и удобства в его использовании. Система сигналов и слотов показала себя очень эффективной в проекте с графическим интерфейсом, так как именно благодаря ей в приложении осуществляется связь между пользователем и приложением: при нажатии определенной кнопки запускается определенный процесс, который к ней привязан. Еще эта система было использована для связи между потоками, а также для связи между разными окнами. Всё эта возможно благодаря Qt, который привносит в язык программирования целый ряд своих переменных и функций, который являются очень полезными при создании приложения с графическим интерфейсом.

Благодаря приложению Qt Creator создание графического интерфейса было очень простым, так как оно всё, что пользователь создал в приложении, автоматически переводит в готовый код, после чего программисту уже остается только написать функционал для созданных объектов. А благодаря приложению Linguist приложение можно переводить на любой существующий язык.

Так как подавляющее число фотографий в современном мире является цифровым, то без фоторедакторов сейчас никуда. Благодаря фоторедакторам можно подправлять неудачные фотографии, удалять с них ненужные объекты или, наоборот, добавлять что-то недостающее.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Объектно-ориентированное программирование на языке C++: учеб. пособие / Ю. А. Луцик, В. Н. Комличенко. – Минск : БГУИР, 2008.

[2] Qt 6 Documentation [Электронный ресурс]. – Электронные данные. – Режим доступа: https://doc.qt.io/qt6 – Дата доступа: 10.12.2023

[3] OpenCV Documentation [Электронный ресурс]. – Электронные данные. – Режим доступа: https://docs.opencv.org/4.x – Дата доступа: 10.12.2023

[4] Qt Creator [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.qt.io/product/development-tools – Дата доступа: 10.12.2023

[5] Adobe Lightroom [Электронный ресурс]. – Электронные данные. – Режим доступа: https://lightroom.adobe.com/ – Дата доступа: 10.12.2023

[6] Microsoft Photos [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/Фотографии_(Microsoft) – Дата доступа: 10.12.2023

[7] GNU Image Manipulation Program [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.gimp.org/ – Дата доступа: 10.12.2023

[8] PhotoFiltre [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.photofiltre-studio.com/ – Дата доступа: 10.12.2023

[9] Adobe Photoshop [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.adobe.com/ru/products/photoshop.html – Дата доступа: 10.12.2023

ПРИЛОЖЕНИЕ А
(Обязательно)
Диаграмма классов

ПРИЛОЖЕНИЕ Б

(Обязательное)

Схема метода void MainWindow::start_proc(QString &)

ПРИЛОЖЕНИЕ В

(Обязательное)

Схема метода void FilterName_window::save_filter_name()

ПРИЛОЖЕНИЕ Г
(Обязательное)
Полный код программы

Файл iconautosizepushbutton.h:

```
#ifndef ICONAUTOSIZEPUSHBUTTON_H
#define ICONAUTOSIZEPUSHBUTTON_H
#include <QPushButton>
#include <QPixmap>
#include <QResizeEvent>
class IconautosizePushButton : public QPushButton {
  Q_OBJECT
public:
  explicit    IconautosizePushButton(QWidget    *parent    =    0)    :
QPushButton(parent) { }
  void set_image_path(QString &s) {
    image_path = s; }
  QString &get_image_path() {
    return image_path; }
public slots:
  void resizeEvent(QResizeEvent *e) override {
    setIconSize(icon().actualSize(QSize(
        e->size().width() - 10,
        e->size().height() - 10)));
    QPushButton::resizeEvent(e); }
private:
  QString image_path;
};
#endif // ICONAUTOSIZEPUSHBUTTON_H
```

Файл image_filters.cpp:

```
#include "image_filters.h"
std::string Filter::get_filter_name() {
  return name; }
Mat Filter::apply() {
  return image; }
Filter::~Filter() {}
Inverse::Inverse(Mat image) {
  name = "Inverse";
  Mat max255 = image.clone();
  max255 = Scalar(255, 255, 255);
  absdiff(max255, image, this->image); }
Original::Original(Mat image) {
  this->image = image;
  name = "Original"; }
Gray::Gray(Mat image) {
  this->image = image;
  name = "Gray";
  cvtColor(image, this->image, COLOR_BGR2GRAY); }
CustomFilter::CustomFilter(std::string name, Mat image, int br, int co,
int st, int cl, int tmp) {
  this->name = name;
  this->image = image;
  brightness = br;
  contrast = co;
  saturation = st;
  clarity = cl;
  temperature = tmp;
  if (brightness) {
    Oper_brightness oper(brightness, this->image);
```

```
    this->image = oper.exec(); }
  if (contrast) {
    Oper_contrast oper(contrast, this->image);
    this->image = oper.exec(); }
  if (saturation) {
    Oper_saturation oper(saturation, this->image);
    this->image = oper.exec(); }
  if (clarity) {
    Oper_clarity oper(clarity, this->image);
    this->image = oper.exec(); }
  if (temperature) {
    Oper_temperature oper(temperature, this->image);
    this->image = oper.exec(); } }
int Filter::get_brightness() {
  return brightness; }
int Filter::get_contrast() {
  return contrast; }
int Filter::get_saturation() {
  return saturation; }
int Filter::get_clarity() {
  return clarity; }
int Filter::get_temperature() {
  return temperature; }
```

Файл image_filters.h:

```
#ifndef FILTERS_H
#define FILTERS_H
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include "image_processing.h"
using namespace cv;
class Filter {
protected:
  Mat image;
  std::string name;
  int brightness, contrast, saturation, clarity, temperature;
public:
  Mat apply();
  std::string get_filter_name();
  int get_brightness();
  int get_contrast();
  int get_saturation();
  int get_clarity();
  int get_temperature();
  virtual ~Filter() = 0;
};
class Inverse : public Filter {
public:
  Inverse(Mat);
};
class Original : public Filter {
public:
  Original(Mat);
};
class Gray : public Filter {
public:
  Gray(Mat);
};
class CustomFilter : public Filter {
public:
  CustomFilter(std::string, Mat, int, int, int, int, int);
```

```
};
#endif
```

Файл image_processing.cpp:

```
#include "image_processing.h"
Mat Oper_brightness::exec() {
  image.convertTo(image, -1, 1, value);
  return image; }
Mat Oper_contrast::exec() {
  double alpha = 1;
  alpha += value / 100.0;
  image.convertTo(image, -1, alpha, 0);
  return image; }
Mat Oper_saturation::exec() {
  cvtColor(image, image, COLOR_BGR2HSV);
  std::vector<Mat> channels;
  split(image, channels);
  double alpha = 1;
  alpha += value / 100.0;
  channels[1].convertTo(channels[1], -1, alpha, 0);
  merge(channels, image);
  cvtColor(image, image, COLOR_HSV2BGR);
  return image; }
Mat Oper_clarity::exec() {
  if (value > 0) {
    double alpha = 1;
    alpha += value / 100.0;
    Mat kernel = (Mat_<double>(3, 3) << 0, -1 * alpha, 0,
                  -1 * alpha, 5 * alpha, -1 * alpha,
                  0, -1 * alpha, 0);
    filter2D(image, image, image.depth(), kernel); }
  return image; }
Mat Oper_temperature::exec() {
  double gamma = 1;
  gamma += abs(value) / 100.0;
  Mat lookUpTable(1, 256, CV_8U);
  uchar *p = lookUpTable.ptr();
  for (int i = 0; i < 256; ++i)
    p[i] = saturate_cast<uchar>(pow(i / 255.0, gamma) * 255.0);
  std::vector<Mat> bgr;
  split(image, bgr);
  if (value < 0)
    LUT(bgr[2], lookUpTable, bgr[2]);
  if (value > 0)
    LUT(bgr[0], lookUpTable, bgr[0]);
  merge(bgr, image);
  return image; }
```

Файл image_processing.h:

```
#ifndef IMAGE_PROCESSING_H
#define IMAGE_PROCESSING_H
#include <QPixmap>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc.hpp>
using namespace cv;
namespace fs = std::filesystem;
typedef Point3_<uint8_t> Pixel;
class Operation {
protected:
  int value;
  Mat image;
```

```
public:
  virtual Mat exec() = 0;
};
class Oper_brightness : public Operation {
public:
  Oper_brightness(int val, Mat image) {
    value = val;
    this->image = image; }
  virtual Mat exec() override;
};
class Oper_contrast : public Operation {
public:
  Oper_contrast(int val, Mat image) {
    value = val;
    this->image = image; }
  virtual Mat exec() override;
};
class Oper_saturation : public Operation {
public:
  Oper_saturation(int val, Mat image) {
    value = val;
    this->image = image; }
  virtual Mat exec() override;
};
class Oper_clarity : public Operation {
public:
  Oper_clarity(int val, Mat image) {
    value = val;
    this->image = image; }
  virtual Mat exec() override;
};
class Oper_temperature : public Operation {
public:
  Oper_temperature(int val, Mat image) {
    value = val;
    this->image = image; }
  virtual Mat exec() override;
};
#endif
```

Файл main.cpp:

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

Файл mainwindow.cpp:

```
#include "mainwindow.h"
MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent) {
  setupUi(this);
  FN_W = new FilterName_window();
  image_info.start_image = new QPixmap;
  image_info.image_in_proc = new QPixmap;
  pixmap = new QGraphicsPixmapItem;
  graphicsScene = new QGraphicsScene;
```

```
        qApp->installTranslator(&qtlangtransl);
        graphicsScene->addItem(pixmap);
        graphicsView_main_im->setScene(graphicsScene);
        image_info.brightness = 0;
        image_info.contrast = 0;
        image_info.saturation = 0;
        image_info.clarity = 0;
        image_info.temperture = 0;
        image_info.filter = FILTER::ORIGINAL;
        FN_W->hide();
        FN_W->setWindowModality(Qt::WindowModality::ApplicationModal);
        graphicsView_main_im->hide();
        pushButton_Rec_open_1->hide();
        pushButton_Rec_open_2->hide();
        pushButton_Rec_open_3->hide();
        pushButton_Rec_open_4->hide();
        pushButton_Rec_open_5->hide();
        pushButton_brightness->hide();
        pushButton_clarity->hide();
        pushButton_temperature->hide();
        pushButton_contrast->hide();
        pushButton_saturation->hide();
        regulation->hide();
        out_amount->hide();
        pushButton_left->hide();
        pushButton_right->hide();
        actionExport->setEnabled(false);
        actionNew_image->setEnabled(false);
        actionEnglish->setEnabled(false);
        pushButton_applyFilter->hide();
        pushButton_filters->hide();
        pushButton_centerF->hide();
        pushButton_leftF->hide();
        pushButton_rightF->hide();
        pushButton_toLeft->hide();
        pushButton_toRight->hide();
        label_centerFname->hide();
        label_leftFname->hide();
        label_rightFname->hide();
        pushButton_back->hide();
        pushButton_addF->hide();
        pushButton_deleteF->hide();
        set_curr_proc(PROCESSES::NON);
        set_rec_opened_butts();
        mythread = new MyThread();
        QObject::connect(QApplication::instance(), &QApplication::aboutToQuit,
mythread, &MyThread::terminateThread);
        mythread->start();
        QObject::connect(mythread,         &MyThread::signalGUI,         this,
&MainWindow::change_image);
        set_connections(); }
      MainWindow::~MainWindow() {
        if (!filters.empty())
          save_filters();
        delete image_info.start_image;
        delete image_info.image_in_proc;
        delete pixmap;
        delete graphicsScene; }
      void MainWindow::change_image(cv::Mat cv_im) {
        QPixmap Qpixmap = QPixmap::fromImage(QtOcv::mat2Image(cv_im));
        pixmap->setPixmap(Qpixmap);
        QSize sz = Qpixmap.size();
```

```
        graphicsView_main_im->fitInView(pixmap, Qt::KeepAspectRatio); }
      void MainWindow::start_proc(QString &QPath) {
        QString Qpath_from;
        if (QPath.isEmpty()) {
          Qpath_from  =  QFileDialog::getOpenFileName(this,  QObject::tr("Open
file"), "", QObject::tr("Images (*.png *.jpg)"), nullptr);
          for (int i = 0; i < Qpath_from.size(); i++) {
            if (Qpath_from[i] == '/')
              Qpath_from[i] = '\\'; }
          if (Qpath_from.isEmpty())
            return; }
        else
          Qpath_from = QPath;
        if (!Qpath_from.isEmpty()) {
          QPixmap Qpixmap(Qpath_from);
          *(image_info.start_image) = Qpixmap;
          *(image_info.image_in_proc) = Qpixmap;
          graphicsView_main_im->show();
          pixmap = new QGraphicsPixmapItem;
          graphicsScene = new QGraphicsScene;
          graphicsScene->addItem(pixmap);
          graphicsView_main_im->setScene(graphicsScene);
          pixmap->setPixmap(Qpixmap);
          graphicsView_main_im->fitInView(pixmap, Qt::KeepAspectRatio);
          label_greeting->hide();
          pushButton_Rec_open_1->hide();
          pushButton_Rec_open_2->hide();
          pushButton_Rec_open_3->hide();
          pushButton_Rec_open_4->hide();
          pushButton_Rec_open_5->hide();
          pushButton_New->hide();
          pushButton_brightness->show();
          pushButton_clarity->show();
          pushButton_temperature->show();
          pushButton_contrast->show();
          pushButton_saturation->show();
          pushButton_left->show();
          pushButton_right->show();
          pushButton_filters->show();
          actionExport->setEnabled(true);
          actionNew_image->setEnabled(true);
          QFile file;

file.setFileName("D:\\University\\cs\\sem3\\cursach\\photored\\recently_opene
d.json");
          file.open(QIODevice::ReadWrite);
          QString s;
          s = file.readAll();
          QJsonDocument d;
          d = QJsonDocument::fromJson(s.toUtf8());
          QJsonArray pathes = d.array();
          QJsonValue path(Qpath_from);
          if (!pathes.contains(path)) {
            int n = pathes.size();
            if (n < 5) {
              pathes.push_back(path); }
            else {
              pathes.pop_front();
              pathes.push_back(path); }
            d.setArray(pathes);
            s = d.toJson();
            file.resize(0);
```

```
            file.write(s.toUtf8()); }
          file.close(); } }
      void MainWindow::main_proc(int value) {
        PROCESSES proc = get_curr_proc();
        QPixmap image = (*image_info.image_in_proc);
        switch (proc) {
        case PROCESSES::BRIGHTNESS: {
          mythread->push(new Oper_brightness(value,
QtOcv::image2Mat(image.toImage())));
          break; }
        case PROCESSES::CONTRAST: {
          mythread->push(new Oper_contrast(value,
QtOcv::image2Mat(image.toImage())));
          break; }
        case PROCESSES::SATURATUIN: {
          mythread->push(new Oper_saturation(value,
QtOcv::image2Mat(image.toImage())));
          break; }
        case PROCESSES::CLARITY: {
          mythread->push(new Oper_clarity(value,
QtOcv::image2Mat(image.toImage())));
          break; }
        case PROCESSES::TEMPERATURE: {
          mythread->push(new Oper_temperature(value,
QtOcv::image2Mat(image.toImage())));
          break; } } }
      void MainWindow::end_main_proc() {
        PROCESSES proc = get_curr_proc();
        int value = regulation->value();
        switch (proc) {
        case PROCESSES::BRIGHTNESS: {
          image_info.brightness = value;
          break; }
        case PROCESSES::CONTRAST: {
          image_info.contrast = value;
          break; }
        case PROCESSES::SATURATUIN: {
          image_info.saturation = value;
          break; }
        case PROCESSES::CLARITY: {
          image_info.clarity = value;
          break; }
        case PROCESSES::TEMPERATURE: {
          image_info.temperture = value;
          break; } } }
      void MainWindow::save_image() {
        QString filter = "PNG(*.png);;JPG(*.jpg)", selected_filter;
        QString filename;
        filename = filename.toUtf8();
        filename = QFileDialog::getSaveFileName(this, QObject::tr("Save File"),
"D:\\", filter, &selected_filter);
        if (!filename.isEmpty()) {
          for (int i = 0; i < filename.size(); i++)
            if (filename[i] == '/')
              filename[i] = '\\';
          prepare_image();
          image_info.image_in_proc->save(filename); } }
      void MainWindow::set_new_image() {
        current_process = PROCESSES::NON;
        show_pressed_button();
        image_info.brightness = 0;
        image_info.contrast = 0;
```

```
    image_info.saturation = 0;
    image_info.clarity = 0;
    image_info.temperture = 0;
    graphicsView_main_im->hide();
    regulation->hide();
    pushButton_brightness->hide();
    pushButton_contrast->hide();
    pushButton_saturation->hide();
    pushButton_clarity->hide();
    pushButton_temperature->hide();
    out_amount->hide();
    pushButton_left->hide();
    pushButton_right->hide();
    pushButton_filters->hide();
    pushButton_toLeft->hide();
    pushButton_leftF->hide();
    label_leftFname->hide();
    pushButton_centerF->hide();
    label_centerFname->hide();
    pushButton_rightF->hide();
    label_rightFname->hide();
    pushButton_toRight->hide();
    pushButton_addF->hide();
    pushButton_deleteF->hide();
    pushButton_back->hide();
    pushButton_applyFilter->hide();
    pushButton_New->show();
    pushButton_Rec_open_1->show();
    pushButton_Rec_open_2->show();
    pushButton_Rec_open_3->show();
    pushButton_Rec_open_4->show();
    pushButton_Rec_open_5->show();
    label_greeting->show();
    actionExport->setEnabled(false);
    actionNew_image->setEnabled(false);
    set_rec_opened_butts();
    delete pixmap;
    delete graphicsScene; }
  void MainWindow::apply_filter() {
    std::string filter_name = filters[filter_number]->get_filter_name();
    if (filter_name == "Inverse") {
      image_info.filter = FILTER::INVERSE; }
    else if (filter_name == "Gray") {
      image_info.filter = FILTER::GRAY; }
    else if (filter_name == "Original") {
      image_info.filter = FILTER::ORIGINAL;
      image_info.brightness = 0;
      image_info.contrast = 0;
      image_info.saturation = 0;
      image_info.clarity = 0;
      image_info.temperture = 0; }
    else {
      image_info.filter = FILTER::CUSTOM;
      image_info.brightness = filters[filter_number]->get_brightness();
      image_info.contrast = filters[filter_number]->get_contrast();
      image_info.saturation = filters[filter_number]->get_saturation();
      image_info.clarity = filters[filter_number]->get_clarity();
      image_info.temperture = filters[filter_number]->get_temperature(); }
    prepare_image();
    back_from_filters(); }
  void MainWindow::rotate_left() {
    Mat image = QtOcv::image2Mat(image_info.image_in_proc->toImage());
```

```
        Mat start_image = QtOcv::image2Mat(image_info.start_image->toImage());
        rotate(image, image, ROTATE_90_COUNTERCLOCKWISE);
        rotate(start_image, start_image, ROTATE_90_COUNTERCLOCKWISE);
        (*image_info.image_in_proc)=QPixmap::fromImage(
QtOcv::mat2Image(image));
        (*image_info.start_image)=QPixmap::fromImage(
QtOcv::mat2Image(start_image));
        change_image(image); }
      void MainWindow::rotate_right() {
        Mat image = QtOcv::image2Mat(image_info.image_in_proc->toImage());
        Mat start_image = QtOcv::image2Mat(image_info.start_image->toImage());
        rotate(image, image, ROTATE_90_CLOCKWISE);
        rotate(start_image, start_image, ROTATE_90_CLOCKWISE);
        (*image_info.image_in_proc)=QPixmap::fromImage(
QtOcv::mat2Image(image));
        (*image_info.start_image)=QPixmap::fromImage(
QtOcv::mat2Image(start_image));
        change_image(image); }
```

Файл mainwindow.h:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include <QFileDialog>
#include <QTranslator>
#include "ui_mainwindow.h"
#include <filesystem>
#include <string>
#include <QJsonArray>
#include <QJsonDocument>
#include <QJsonValue>
#include <QGraphicsScene>
#include <QGraphicsPixmapItem>
#include "workerthread.h"
#include "cvmatandqimage.h"
#include <opencv2/core/core.hpp>
#include "MyRing.h"
#include "image_filters.h"
#include "set_FilterName_window.h"
enum class PROCESSES {
  BRIGHTNESS,
  CONTRAST,
  SATURATUIN,
  CLARITY,
  TEMPERATURE,
  ROTATION,
  FILTER,
  NON
};
enum class FILTER {
  INVERSE,
  ORIGINAL,
  GRAY,
  CUSTOM
};
namespace fs = std::filesystem;
class MyThread;
class MainWindow : public QMainWindow, protected Ui::MainWindow {
  Q_OBJECT
  MyThread *mythread;
  PROCESSES current_process;
  MyRing<Filter *> filters;
```

```
  int filter_number;
  void set_connections();
  QTranslator qtlangtransl;
public:
  FilterName_window *FN_W;
  explicit MainWindow(QWidget *parent = nullptr);
  void set_curr_proc(PROCESSES);
  PROCESSES get_curr_proc();
  void prepare_image();
  void set_filters();
  void save_filters();
  cv::Mat get_filtered_image(int);
  std::string get_filter_name(int);
  void set_filter_number(int);
  void next_prev_filter(int);
  ~MainWindow();
public slots:
  void change_image(cv::Mat);
  void set_rec_opened_butts();
  void start_proc(QString &);
  void main_proc(int);
  void set_slider_limits();
  void end_main_proc();
  void rotate_left();
  void rotate_right();
  void save_image();
  void set_new_image();
  void set_filters_buttons();
  void set_deleteF_enabled(std::string);
  void back_from_filters();
  void apply_filter();
  void delete_filter();
  void add_filter();
  void show_pressed_button();
  void change_language(const char *);
protected:
  void changeEvent(QEvent *) override;
  QGraphicsScene *graphicsScene;
  QGraphicsPixmapItem *pixmap;
  struct image_info {
    QPixmap *start_image;
    QPixmap *image_in_proc;
    int brightness, contrast, saturation, clarity, temperture;
    FILTER filter;
  } image_info;
};
#endif // MAINWINDOW_H
```

Файл MyRing.h:

```
#ifndef MYRING_H
#define MYRING_H
template <typename T>
class RingNode {
public:
  T data;
  RingNode *next;
  RingNode *prev;
  RingNode() {
    next = nullptr;
    prev = nullptr; }
  RingNode(T data) {
    this->data = data;
```

```
    next = nullptr;
    prev = nullptr; }
};
template <typename T>
class MyRing {
  RingNode<T> *head;
  int ring_size;
public:
  MyRing() {
    head = nullptr;
    ring_size = 0; }
  MyRing(const MyRing &other) {
    RingNode<T> *temp = other.head;
    for (int i = 0; i < other.ring_size; i++) {
      push(temp->data);
      temp = temp->next; }
    delete temp; }
  ~MyRing() {
    while (ring_size)
      pop_head(); }
  void push(T data) {
    RingNode<T> *newNode = new RingNode<T>(data);
    if (ring_size == 0) {
      head = newNode;
      head->next = head;
      head->prev = head; }
    else {
      newNode->prev = head->prev;
      head->prev->next = newNode;
      newNode->next = head;
      head->prev = newNode; }
    ring_size++; }
  void pop_head() {
    if (ring_size == 0)
      throw "Ring is empty";
    else if (ring_size == 1) {
      delete head;
      head = nullptr; }
    else {
      RingNode<T> *temp = head;
      head->prev->next = head->next;
      head->next->prev = head->prev;
      head = head->next;
      delete temp; }
    ring_size--; }
  T get_data() {
    if (ring_size == 0)
      throw "Ring is empty";
    return head->data; }
  void next_node() {
    if (ring_size == 0)
      throw "Ring is empty";
    head = head->next; }
  void prev_node() {
    if (ring_size == 0)
      throw "Ring is empty";
    head = head->prev; }
  bool empty() {
    if (ring_size)
      return false;
    return true; }
  int size() {
```

```
    return ring_size; }
  void clean() {
    while (ring_size)
      pop_head();
    head = nullptr; }
  T &operator[](const int index) {
    RingNode<T> *temp = head;
    if (index > 0) {
      for (int i = 0; i < index; i++)
        temp = temp->next; }
    if (index < 0) {
      for (int i = 0; i > index; i--)
        temp = temp->prev; }
    return temp->data; }
};
#endif
```

Файл set_FilterName_window.cpp:

```
#include "set_FilterName_window.h"
FilterName_window::FilterName_window(QWidget *parent) : QWidget(parent)
{
  setupUi(this);
  QObject::connect(pushButton_cansel, &QPushButton::clicked, this,
&FilterName_window::hide);
  QObject::connect(pushButton_save, &QPushButton::clicked, this,
&FilterName_window::save_filter_name); }
void FilterName_window::set_filters(MyRing<Filter *> *main_filters) {
  filters = main_filters; }
bool FilterName_window::is_name_in_filters(std::string filter_name) {
  for (int i = 0; i < filters->size(); i++) {
    if ((*filters)[i]->get_filter_name() == filter_name)
      return true; }
  return false; }
void FilterName_window::save_filter_name() {
  label_2->setText("");
  std::string s = lineEdit->text().toStdString();
  while (s[0] == ' ')
    s.erase(0, 1);
  while (s[s.length() - 1] == ' ')
    s.erase(s.length() - 1, 1);
  if (s.empty()) {
    label_2->setText(tr("Name is empty"));
    return; }
  if (is_name_in_filters(s)) {
    label_2->setText(tr("This name already exists"));
    return; }
  filter_name = s;
  emit filter_name_got();
  hide(); }
std::string FilterName_window::get_filter_name() {
  return filter_name; }
void FilterName_window::changeEvent(QEvent *e) {
  if (e->type() == QEvent::LanguageChange) {
    retranslateUi(this); } }
```

Файл set_FilterName_window.h:

```
#ifndef SET_FILTERNAME_WINDOW_H
#define SET_FILTERNAME_WINDOW_H
#include "ui_set_FilterName_window.h"
#include "MyRing.h"
#include "image_filters.h"
```

```
#include <QString>
class FilterName_window : public QWidget, protected Ui::Form {
  Q_OBJECT
  MyRing<Filter *> *filters;
  std::string filter_name;
public:
  explicit FilterName_window(QWidget *parent = nullptr);
  void set_filters(MyRing<Filter *> *);
  bool is_name_in_filters(std::string);
  std::string get_filter_name();
protected:
  void changeEvent(QEvent *) override;
public slots:
  void save_filter_name();
signals:
  void filter_name_got();
};
#endif
```

Файл utils.cpp:

```
#include <QJsonArray>
#include <QJsonDocument>
#include <QJsonValue>
#include <QJsonObject>
#include "mainwindow.h"
#include <QString>
#include <QFile>
void MainWindow::set_rec_opened_butts() {
  QFile file;

file.setFileName("D:\\University\\cs\\sem3\\cursach\\photored\\recently_opene
d.json");
  if (file.open(QIODevice::ReadOnly)) {
    QString s;
    s = file.readAll();
    file.close();
    QJsonDocument d;
    d = QJsonDocument::fromJson(s.toUtf8());
    QJsonArray pathes = d.array();
    QJsonValue path;
    int n = pathes.size(), button_n = 0;
    for (int i = 0; i < n; i++) {
      path = pathes[i];
      s = path.toString();
      QPixmap im(s);
      if (!im.isNull()) {
        QIcon but_im(im);
        switch (button_n) {
        case 0:
          pushButton_Rec_open_1->set_image_path(s);
          pushButton_Rec_open_1->setIcon(but_im);
          pushButton_Rec_open_1->show();
          break;
        case 1:
          pushButton_Rec_open_2->set_image_path(s);
          pushButton_Rec_open_2->setIcon(but_im);
          pushButton_Rec_open_2->show();
          break;
        case 2:
          pushButton_Rec_open_3->set_image_path(s);
          pushButton_Rec_open_3->setIcon(but_im);
          pushButton_Rec_open_3->show();
```

```
                break;
              case 3:
                pushButton_Rec_open_4->set_image_path(s);
                pushButton_Rec_open_4->setIcon(but_im);
                pushButton_Rec_open_4->show();
                break;
              case 4:
                pushButton_Rec_open_5->set_image_path(s);
                pushButton_Rec_open_5->setIcon(but_im);
                pushButton_Rec_open_5->show();
                break; }
              button_n++; } } } }
      void MainWindow::set_curr_proc(PROCESSES n) {
        current_process = n; }
      PROCESSES MainWindow::get_curr_proc() {
        return current_process; }
      void MainWindow::set_slider_limits() {
        PROCESSES n = get_curr_proc();
        if (n == PROCESSES::CLARITY) {
          regulation->setMinimum(0);
          regulation->setMaximum(100); }
        else {
          regulation->setMinimum(-100);
          regulation->setMaximum(100); }
        switch (n) {
        case PROCESSES::BRIGHTNESS: {
          regulation->setValue(image_info.brightness);
          out_amount->setNum(image_info.brightness);
          break; }
        case PROCESSES::CONTRAST: {
          regulation->setValue(image_info.contrast);
          out_amount->setNum(image_info.contrast);
          break; }
        case PROCESSES::SATURATUIN: {
          regulation->setValue(image_info.saturation);
          out_amount->setNum(image_info.saturation);
          break; }
        case PROCESSES::CLARITY: {
          regulation->setValue(image_info.clarity);
          out_amount->setNum(image_info.clarity);
          break; }
        case PROCESSES::TEMPERATURE: {
          regulation->setValue(image_info.temperture);
          out_amount->setNum(image_info.temperture);
          break; } } }
      void MainWindow::prepare_image() {
        QPixmap image = *(image_info.start_image);
        if (current_process != PROCESSES::BRIGHTNESS) {
          Oper_brightness oper(image_info.brightness,
QtOcv::image2Mat(image.toImage()));
          image = QPixmap::fromImage(QtOcv::mat2Image(oper.exec())); }
        if (current_process != PROCESSES::CONTRAST) {
          Oper_contrast oper(image_info.contrast,
QtOcv::image2Mat(image.toImage()));
          image = QPixmap::fromImage(QtOcv::mat2Image(oper.exec())); }
        if (current_process != PROCESSES::SATURATUIN) {
          Oper_saturation oper(image_info.saturation,
QtOcv::image2Mat(image.toImage()));
          image = QPixmap::fromImage(QtOcv::mat2Image(oper.exec())); }
        if (current_process != PROCESSES::CLARITY) {
          Oper_clarity oper(image_info.clarity,
QtOcv::image2Mat(image.toImage()));
```

```
          image = QPixmap::fromImage(QtOcv::mat2Image(oper.exec())); }
        if (current_process != PROCESSES::TEMPERATURE) {
          Oper_temperature oper(image_info.temperture,
QtOcv::image2Mat(image.toImage()));
          image = QPixmap::fromImage(QtOcv::mat2Image(oper.exec())); }
        if (image_info.filter == FILTER::INVERSE) {
          Inverse filt(QtOcv::image2Mat(image.toImage()));
          image = QPixmap::fromImage(QtOcv::mat2Image(filt.apply())); }
        else if (image_info.filter == FILTER::GRAY) {
          Gray filt(QtOcv::image2Mat(image.toImage()));
          image = QPixmap::fromImage(QtOcv::mat2Image(filt.apply())); }
        (*image_info.image_in_proc) = image; }
      void MainWindow::set_filter_number(int n) {
        filter_number = n; }
      void MainWindow::set_filters() {
        pushButton_deleteF->setEnabled(false);
        FILTER temp = image_info.filter;
        image_info.filter = FILTER::CUSTOM;
        prepare_image();
        image_info.filter = temp;
        filters.clean();
        Mat image = QtOcv::image2Mat(image_info.image_in_proc->toImage());
        Mat start_image = QtOcv::image2Mat(image_info.start_image->toImage());
        filters.push(new Inverse(image));
        filters.push(new Original(start_image.clone()));
        filters.push(new Gray(image));
        QFile file;

file.setFileName("D:\\University\\cs\\sem3\\cursach\\photored\\filters_inform
.json");
        if (file.open(QIODevice::ReadOnly)) {
          QString s;
          s = file.readAll();
          file.close();
          QJsonDocument d;
          d = QJsonDocument::fromJson(s.toUtf8());
          QJsonArray js_filters = d.array();
          QJsonObject js_filter;
          std::string filter_name;
          int br, co, st, cl, tmp;
          for (int i = 0; i < js_filters.size(); i++) {
            js_filter = js_filters[i].toObject();
            filter_name = js_filter["name"].toString().toStdString();
            br = js_filter["brightness"].toInt();
            co = js_filter["contrast"].toInt();
            st = js_filter["saturation"].toInt();
            cl = js_filter["clarity"].toInt();
            tmp = js_filter["temperature"].toInt();
            filters.push(new  CustomFilter(filter_name,  start_image.clone(),
br, co, st, cl, tmp)); } } }
      void MainWindow::save_filters() {
        QFile file;

file.setFileName("D:\\University\\cs\\sem3\\cursach\\photored\\filters_inform
.json");
        file.open(QIODevice::WriteOnly);
        file.resize(0);
        QJsonObject js_filter;
        QJsonArray js_filters;
        QJsonDocument d;
        for (int i = 0; i < filters.size(); i++) {
          std::string filter_name = filters[i]->get_filter_name();
```

```
          if (filter_name != "Inverse" && filter_name != "Original" &&
filter_name != "Gray") {
            js_filter["name"] = QString::fromStdString(filter_name);
            js_filter["brightness"] = filters[i]->get_brightness();
            js_filter["contrast"] = filters[i]->get_contrast();
            js_filter["saturation"] = filters[i]->get_saturation();
            js_filter["clarity"] = filters[i]->get_clarity();
            js_filter["temperature"] = filters[i]->get_temperature();
            js_filters.push_back(js_filter); } }
        d.setArray(js_filters);
        QString s = d.toJson();
        file.write(s.toUtf8());
        file.close(); }
      void MainWindow::set_filters_buttons() {
        QIcon            leftF(QPixmap::fromImage(QtOcv::mat2Image(filters[0]-
>apply())));
        pushButton_leftF->setIcon(leftF);
        pushButton_leftF->setIconSize(pushButton_leftF-
>icon().actualSize(QSize(pushButton_leftF->size().width()        -        10,
pushButton_leftF->size().height() - 10)));
        label_leftFname->setText(QString::fromStdString(filters[0]-
>get_filter_name()));
        QIcon          centerF(QPixmap::fromImage(QtOcv::mat2Image(filters[1]-
>apply())));
        pushButton_centerF->setIcon(centerF);
        pushButton_centerF->setIconSize(pushButton_centerF-
>icon().actualSize(QSize(pushButton_centerF->size().width()       -       10,
pushButton_centerF->size().height() - 10)));
        label_centerFname->setText(QString::fromStdString(filters[1]-
>get_filter_name()));
        QIcon           rightF(QPixmap::fromImage(QtOcv::mat2Image(filters[2]-
>apply())));
        pushButton_rightF->setIcon(rightF);
        pushButton_rightF->setIconSize(pushButton_rightF-
>icon().actualSize(QSize(pushButton_rightF->size().width()        -        10,
pushButton_rightF->size().height() - 10)));
        label_rightFname->setText(QString::fromStdString(filters[2]-
>get_filter_name())); }
      void MainWindow::next_prev_filter(int n) {
        if (n > 0)
          filters.next_node();
        if (n < 0)
          filters.prev_node(); }
      Mat MainWindow::get_filtered_image(int n) {
        return filters[n]->apply(); }
      std::string MainWindow::get_filter_name(int n) {
        return filters[n]->get_filter_name(); }
      void MainWindow::set_deleteF_enabled(std::string filter_name) {
        if (filter_name == "Inverse" || filter_name == "Original" || filter_name
== "Gray")
          pushButton_deleteF->setEnabled(false);
        else
          pushButton_deleteF->setEnabled(true); }
      void MainWindow::back_from_filters() {
        prepare_image();
        change_image(QtOcv::image2Mat((*image_info.image_in_proc).toImage()));
        save_filters();
        filters.clean();
        pushButton_toLeft->hide();
        pushButton_leftF->hide();
        label_leftFname->hide();
        pushButton_centerF->hide();
```

```
        label_centerFname->hide();
        pushButton_rightF->hide();
        label_rightFname->hide();
        pushButton_toRight->hide();
        pushButton_addF->hide();
        pushButton_deleteF->hide();
        pushButton_back->hide();
        pushButton_applyFilter->hide();
        pushButton_brightness->show();
        pushButton_contrast->show();
        pushButton_saturation->show();
        pushButton_clarity->show();
        pushButton_temperature->show();
        pushButton_filters->show();
        pushButton_left->show();
        pushButton_right->show(); }
      void MainWindow::delete_filter() {
        switch (filter_number) {
        case 0: {
          filters.pop_head();
          break; }
        case 1: {
          filters.next_node();
          filters.pop_head();
          filters.prev_node();
          break; }
        case 2: {
          filters.next_node();
          filters.next_node();
          filters.pop_head();
          filters.prev_node();
          filters.prev_node();
          break; } }
        set_filters_buttons();
        prepare_image();
        change_image(QtOcv::image2Mat((*image_info.image_in_proc).toImage()));
}
      void MainWindow::add_filter() {
        std::string filter_name = FN_W->get_filter_name();
        filters.push(new                                CustomFilter(filter_name,
QtOcv::image2Mat((*image_info.start_image).toImage()),  image_info.brightness,
image_info.contrast,         image_info.saturation,         image_info.clarity,
image_info.temperture)); }
      void MainWindow::show_pressed_button() {
        QString pressed = "background-color: gray";
        QString not_pressed = "";
        if (current_process == PROCESSES::BRIGHTNESS)
          pushButton_brightness->setStyleSheet(pressed);
        else
          pushButton_brightness->setStyleSheet(not_pressed);
        if (current_process == PROCESSES::CONTRAST)
          pushButton_contrast->setStyleSheet(pressed);
        else
          pushButton_contrast->setStyleSheet(not_pressed);
        if (current_process == PROCESSES::SATURATUIN)
          pushButton_saturation->setStyleSheet(pressed);
        else
          pushButton_saturation->setStyleSheet(not_pressed);
        if (current_process == PROCESSES::CLARITY)
          pushButton_clarity->setStyleSheet(pressed);
        else
          pushButton_clarity->setStyleSheet(not_pressed);
```

```
        if (current_process == PROCESSES::TEMPERATURE)
          pushButton_temperature->setStyleSheet(pressed);
        else
          pushButton_temperature->setStyleSheet(not_pressed); }
      void MainWindow::set_connections() {
        QObject::connect(pushButton_New, &QPushButton::clicked, this, [&]()
                         { QString tmp; start_proc(tmp); });
        QObject::connect(pushButton_Rec_open_1,  &QPushButton::clicked,  this,
[&](){start_proc(pushButton_Rec_open_1->get_image_path()); });
        QObject::connect(pushButton_Rec_open_2,  &QPushButton::clicked,  this,
[&](){start_proc(pushButton_Rec_open_2->get_image_path()); });
        QObject::connect(pushButton_Rec_open_3,  &QPushButton::clicked,  this,
[&](){start_proc(pushButton_Rec_open_3->get_image_path()); });
        QObject::connect(pushButton_Rec_open_4,  &QPushButton::clicked,  this,
[&](){start_proc(pushButton_Rec_open_4->get_image_path()); });
        QObject::connect(pushButton_Rec_open_5,  &QPushButton::clicked,  this,
[&](){start_proc(pushButton_Rec_open_5->get_image_path()); });
        QObject::connect(pushButton_brightness,  &QPushButton::clicked,  this,
[&]()                       {  end_main_proc();
                               set_curr_proc(PROCESSES::BRIGHTNESS);
                               show_pressed_button();
                               set_slider_limits();
                               prepare_image(); });
        QObject::connect(pushButton_contrast,   &QPushButton::clicked,   this,
[&]()                       {  end_main_proc();
                               set_curr_proc(PROCESSES::CONTRAST);
                               show_pressed_button();
                               set_slider_limits();
                               prepare_image(); });
        QObject::connect(pushButton_saturation,  &QPushButton::clicked,  this,
[&]()                       {  end_main_proc();
                               set_curr_proc(PROCESSES::SATURATUIN);
                               show_pressed_button();
                               set_slider_limits();
                               prepare_image(); });
        QObject::connect(pushButton_clarity,    &QPushButton::clicked,   this,
[&]()                       {  end_main_proc();
                               set_curr_proc(PROCESSES::CLARITY);
                               show_pressed_button();
                               set_slider_limits();
                               prepare_image(); });
        QObject::connect(pushButton_temperature, &QPushButton::clicked, this,
[&]()                       {  end_main_proc();
                               set_curr_proc(PROCESSES::TEMPERATURE);
                               show_pressed_button();
                               set_slider_limits();
                               prepare_image(); });
        QObject::connect(regulation,   SIGNAL(sliderMoved(int)),   out_amount,
SLOT(setNum(int)));
        QObject::connect(regulation,       &QSlider::sliderMoved,       this,
&MainWindow::main_proc);
        QObject::connect(actionExport, &QAction::triggered, this, [&]()
                         {  end_main_proc();
                               set_curr_proc(PROCESSES::NON);
                               save_image(); });
        QObject::connect(actionNew_image,      &QAction::triggered,      this,
&MainWindow::set_new_image);
        QObject::connect(pushButton_left, &QPushButton::clicked, this, [&]()
                         {  end_main_proc();
                               set_curr_proc(PROCESSES::ROTATION);
                               show_pressed_button();
                               prepare_image();
```

```
                                rotate_left(); });
        QObject::connect(pushButton_right, &QPushButton::clicked, this, [&]()
                         {  end_main_proc();
                            set_curr_proc(PROCESSES::ROTATION);
                            show_pressed_button();
                            prepare_image();
                            rotate_right(); });
        QObject::connect(pushButton_filters,    &QPushButton::clicked,    this,
[&]()                      {  end_main_proc();
                            set_curr_proc(PROCESSES::FILTER);
                            show_pressed_button();
                            set_filters();
                            set_filters_buttons(); });
        QObject::connect(pushButton_toLeft, &QPushButton::clicked, this, [&]()
                         {  next_prev_filter(-1);
                            set_filters_buttons(); });
        QObject::connect(pushButton_toRight,    &QPushButton::clicked,    this,
[&]()                      {  next_prev_filter(1);
                            set_filters_buttons(); });
        QObject::connect(pushButton_leftF, &QPushButton::clicked, this, [&]()
                         { set_filter_number(0);
                            change_image(get_filtered_image(0));
                            set_deleteF_enabled(get_filter_name(0)); });
        QObject::connect(pushButton_centerF,    &QPushButton::clicked,    this,
[&]()                      { set_filter_number(1);
                            change_image(get_filtered_image(1));
                            set_deleteF_enabled(get_filter_name(1)); });
        QObject::connect(pushButton_rightF, &QPushButton::clicked, this, [&]()
                         { set_filter_number(2);
                            change_image(get_filtered_image(2));
                            set_deleteF_enabled(get_filter_name(2)); });
        QObject::connect(pushButton_back,     &QPushButton::clicked,     this,
&MainWindow::back_from_filters);
        QObject::connect(pushButton_applyFilter,  &QPushButton::clicked,  this,
&MainWindow::apply_filter);
        QObject::connect(pushButton_deleteF,    &QPushButton::clicked,    this,
&MainWindow::delete_filter);
        QObject::connect(pushButton_addF, &QPushButton::clicked, this, [&]()
                         {   FN_W->set_filters(&filters);
                             FN_W->show(); });
        QObject::connect(FN_W,    &FilterName_window::filter_name_got,    this,
&MainWindow::add_filter);
        QObject::connect(actionRussian, &QAction::triggered, this, [&]()
                         { change_language("ru");
                            actionRussian->setEnabled(false);
                            actionEnglish->setEnabled(true);
                            actionBelarusian->setEnabled(true); });
        QObject::connect(actionEnglish, &QAction::triggered, this, [&]()
                         { change_language("en");
                            actionRussian->setEnabled(true);
                            actionEnglish->setEnabled(false);
                            actionBelarusian->setEnabled(true); });
        QObject::connect(actionBelarusian, &QAction::triggered, this, [&]()
                         { change_language("be");
                            actionRussian->setEnabled(true);
                            actionEnglish->setEnabled(true);
                            actionBelarusian->setEnabled(false); }); }
      void MainWindow::changeEvent(QEvent *e) {
        if (e->type() == QEvent::LanguageChange) {
          retranslateUi(this); } }
      void MainWindow::change_language(const char *lng) {
        if (!qtlangtransl.load("photored_" + QString(lng) + ".")) { } }
```

Файл viewwithoutwheel.h:

```
#ifndef VIEWWITHOUTWHEEL_H
#define VIEWWITHOUTWHEEL_H
#include <QGraphicsView>
class ViewWithoutWheel : public QGraphicsView {
  Q_OBJECT
public:
  explicit ViewWithoutWheel(QWidget *parent = 0) : QGraphicsView(parent)
{ }
public slots:
  virtual void wheelEvent(QWheelEvent *e) override { }
  virtual void resizeEvent(QResizeEvent *e) override {
    QGraphicsView::resizeEvent(e);
    if (scene() != nullptr)
      fitInView(scene()->items()[0], Qt::KeepAspectRatio); }};
#endif
```

Файл workerthread.cpp:

```
#include "workerthread.h"
void MyThread::run() {
  while (!isInterruptionRequested()) {
    while (queue.empty()) {
      if (isInterruptionRequested())
        return;
      msleep(10); }
    Operation *oper_now = queue.front();
    queue.pop();
    cv::Mat image = oper_now->exec();
    emit signalGUI(image);
    delete oper_now; } }
void MyThread::push(Operation *oper) {
  queue.push(oper); }
```

Файл workerthread.h:

```
#ifndef WORKERTHREAD_H
#define WORKERTHREAD_H
#include <QThread>
#include <queue>
#include "mainwindow.h"
#include "image_processing.h"
#include <opencv2/core/core.hpp>
class MainWindow;
class Operation;
class MyThread : public QThread {
  Q_OBJECT
public:
  std::queue<Operation *> queue;
  MyThread() {}
  void push(Operation *);
protected:
  virtual void run() override;
signals:
  void signalGUI(cv::Mat);
public slots:
  void terminateThread() {
    if (isRunning()) {
      requestInterruption();
      wait(); } }
};
#endif
```

ПРИЛОЖЕНИЕ Д
(Обязательное)
Ведомость документов