Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Операционные системы и системное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

Файловый менеджер с расширенным функционалом для Windows

БГУИР КП 1–40 02 01 225 ПЗ

Студент: группы 250502,   
Потейчук О.В.

Руководитель: ассистент каф. ЭВМ Басак Д. В.

Минск 2024

Учреждение образования

«Белорусский Учреждение образования

«Белорусский государственный университет информатики   
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой

*––––––––––––––––––––––––*

(подпись)

––––––––––––––––––2024 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту     *Потейчуку Олегу Валентиновичу –––––––––––––*

Тема проекта\_\_\_\_\_\_\_*Файловый менеджер с расширенным функционалом для Windows\_*  –––––––––––––––––––––––                  ––   ––––

2. Срок сдачи студентом законченного проекта––––*10 мая 2023 г. --------------*

3. Исходные данные к проекту*-пЯзык программирования – С++, среда разработки – Qt-Creator -------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------*

*––––––––––––––––––––––––––––––––––––––––––––––––––––––           —––--------*

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

1. Лист задания.

2. Введение.

3. Обзор литературы.

3.1. Обзор методов и алгоритмов решения поставленной задачи.

4. Функциональное проектирование.

4.1. Структура входных и выходных данных.

4.2. Разработка диаграммы классов.

4.3. Описание классов.

5. Разработка программных модулей.

5.1. Разработка схем алгоритмов(два наиболее важных метода).

5.2. Разработка алгоритмов (описание алгоритмов по шагам, для двух методов).

6. Результаты работы.

7. Заключение

8. Литература

9. Приложения

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

*1. Диаграмма классов. ––––––––––––––––––––––––––––––*

*2. Схема алгоритма копирования единичного файла*

*3. Схема алгоритма обработки отпускания перетаскиваемых элементов*

*4. Схема структурная*

6. Консультант по проекту (с обозначением разделов проекта) Д.В. Басак

7. Дата выдачи задания –––––*23.02.2024г.––––––––––––––––––––––   –*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

*1. Выбор задания. Разработка содержания пояснительной записки. Перечень графического материала – 15 %;––––––––––––––––––––––––––––*

*разделы 2, 3 – 10 %;–––––––––––––––––––––––––––––––––––––*

*разделы 4 к –20 %;–––––––––––––––––––––––––––––––––––––––*

*разделы 5 к – 35 %;––––––––––––––––––––––––––––––––––––––*

*раздел 6,7,8 – 5 %;–––––––––––––––––––––––––––––––––––––––*

*раздел 9 к – 5%;–––––––––––––––––––––––––––––––––––––––––*

*оформление пояснительной записки и графического материала к 10.04.24 – 10 %*

*Защита курсового проекта с 28.05.24 по 10.06.24г.–––––––––––––––––––––––––*

РУКОВОДИТЕЛЬ Д.В. Басак

(подпись)

Задание принял к исполнению *–––––––\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_––*

(дата и подпись студента)

Содержание

[ВВЕДЕНИЕ 5](#_Toc163269557)

[1 ПОСТАНОВКА ЗАДАЧИ 6](#_Toc163269558)

[2 ОБЗОР ЛИТЕРАТУРЫ 7](#_Toc163269559)

[2.1 Анализ существующих аналогов 7](#_Toc163269560)

[2.2 Требования к работе программы 9](#_Toc163269561)

[3 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 11](#_Toc163269562)

[3.1 Модуль основного окна 11](#_Toc163269563)

[3.2 Модуль файловой системы 11](#_Toc163269564)

[3.3 Модуль отслеживания истории 11](#_Toc163269565)

[3.4 Модуль меню при нажатии ПКМ 12](#_Toc163269566)

[4 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 13](#_Toc163269567)

[4.1 Входные и выходные данные 13](#_Toc163269568)

[4.2 Модуль основного окна 15](#_Toc163269569)

[4.3 Модуль файловой системы 16](#_Toc163269570)

[4.4 Модуль отслеживания истории 17](#_Toc163269571)

[4.5 Модуль меню при нажатии ПКМ 18](#_Toc163269572)

[5 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 20](#_Toc163269573)

[5.1 Алгоритм копирования единичного файла 20](#_Toc163269574)

[5.2 Алгоритм обработки отпускания перетаскиваемых элементов 20](#_Toc163269575)

[6 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ 22](#_Toc163269576)

[6.1 Тестирование ввода некорректного пути в строку ввода 22](#_Toc163269577)

[6.2 Тестирование копирование файла в его же каталог 22](#_Toc163269578)

[6.3 Тестирование попытки переименования файла в уже существующее название 23](#_Toc163269579)

[6.4 Тестирование попытки переноса файла с идентичным названием 24](#_Toc163269580)

[7 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 25](#_Toc163269581)

[7.1 Системные требования 25](#_Toc163269582)

[7.2 Использование приложения 25](#_Toc163269583)

[ЗАКЛЮЧЕНИЕ 29](#_Toc163269584)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 30](#_Toc163269585)

[ПРИЛОЖЕНИЕ А 31](#_Toc163269586)

[ПРИЛОЖЕНИЕ Б 32](#_Toc163269587)

[ПРИЛОЖЕНИЕ В 33](#_Toc163269588)

[ПРИЛОЖЕНИЕ Г 34](#_Toc163269589)

[ПРИЛОЖЕНИЕ Д 35](#_Toc163269590)

ВВЕДЕНИЕ

Файловый менеджер – компьютерная программа, предоставляющая интерфейс пользователю для работы с файловой системой и файлами. Файловый менеджер позволяет выполнять наиболее частые операции над файлами — создание, открытие/проигрывание/просмотр, редактирование, перемещение, переименование, копирование, удаление, изменение атрибутов и свойств, поиск файлов и назначение прав. Помимо основных функций, многие файловые менеджеры включают ряд дополнительных возможностей, например, таких, как работа с сетью.

Существует два вида файловых менеджеров — навигационные и ортодоксальные. Основное отличие — в последних имеется две панели, реализована соответствующая модель работы.

Язык программирования C++ с использованием Qt представляют собой мощную комбинацию инструментов для разработки программного обеспечения, обладающую высокой производительностью и гибкостью. C++ является языком программирования общего назначения, известным своей эффективностью и возможностью создания высокопроизводительных приложений. Qt, в свою очередь, предоставляет богатые возможности для разработки графического интерфейса и работы с различными аспектами приложений. Qt также предлагает обширный набор библиотек и инструментов, включая графические элементы управления, инструменты работы с сетью, базами данных и мультимедиа.

Qt Creator, интегрированная среда разработки, предоставляет удобный набор инструментов для написания, отладки и развертывания приложений, что упрощает процесс работы разработчиков. Благодаря активным сообществам пользователей и обширной документации, разработчики получают поддержку, информацию и ресурсы для изучения и использования C++ с Qt в своих проектах.

# ПОСТАНОВКА ЗАДАЧИ

Цель данного курсового проекта состоит в разработке файлового менеджера для операционной системы Windows. Файловый менеджер представляет собой приложение, предназначенное для управления файлами и папками на диске компьютера. Он должен обладать удобным и интуитивно понятным интерфейсом, позволяющим пользователям выполнять различные операции с файлами, такие как создание, копирование, перемещение, удаление, переименование и просмотр.

Функциональные требования:

* Интерфейс пользователя: разработать удобный и интуитивно понятный интерфейс для взаимодействия с файловым менеджером. Включить в него элементы управления для навигации по файловой системе, а также кнопки и контекстное меню для выполнения операций с файлами и папками.
* Поддержка различных типов файлов: обеспечить возможность работы с различными типами файлов, включая текстовые, графические, аудио- и видеофайлы, а также файлы различных форматов.
* Функции поиска и фильтрации: реализовать функционал для поиска файлов и папок по их названиям, типам и другим атрибутам. Также добавить возможность фильтрации файлов по их типам и размерам.
* Поддержка множественного выделения и операций над ними: обеспечить возможность выделения нескольких файлов и папок одновременно для выполнения операций над ними, таких как копирование, перемещение и удаление.

Возможные проблемы реализации:

* Совместимость с различными версиями Windows: Файловый менеджер должен быть совместим с различными версиями операционной системы Windows.
* Масштабируемость и расширяемость: при проектировании архитектуры файлового менеджера следует учитывать его масштабируемость и возможность расширения функционала в будущем. Это может потребовать использования модульной структуры и обеспечение гибкости при добавлении новых возможностей и функций.

Решение указанных проблем и успешная реализация требует тщательного планирования, анализа требований и использования современных подходов к разработке программного обеспечения.

# ОБЗОР ЛИТЕРАТУРЫ

## Анализ существующих аналогов

Целью выбора темы курсового проекта о создании файлового менеджера на Qt было желание поглубже изучить работу с файловой системой и применить принципы ООП при разработке приложения. В процессе создания приложения будут рассмотрены основные принципы работы с файлами и каталогами через графический интерфейс на платформе Qt. Для того, чтобы создать корректно работающее приложение, необходимо иметь представление об уже реализованных аналогах и изучить содержание источника [3].

### Total Commander

Total Commander – это файловый менеджер для Windows, такой как Проводник в Windows для копирования, перемещения или удаления файлов. Однако Total Commander может делать гораздо больше, чем Проводник, например, распаковывать файлы, получать доступ к ftp-серверам, сравнивать файлы по контенту, имеет в себе встроенный ZIP-совместимый упаковщик и внутренние распаковщики для множества файловых расширений

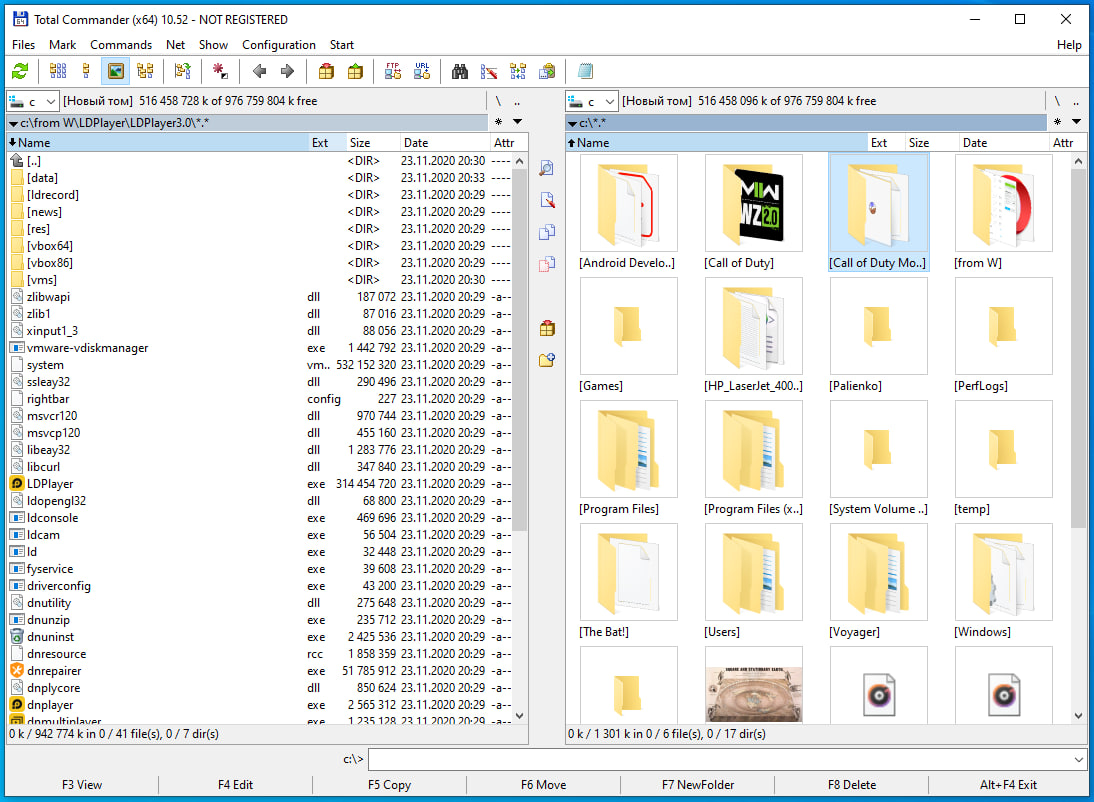


Рисунок 2.1 – Скриншот Total Commander

### Directory Opus

Directory Opus – Коммерческий файловый менеджер, который был впервые запущен в 1990 году. Имеет обширный набор функций и гибкие настройки, за что полюбился многим пользователям. Из разряда бесплатных программ, перекочевал в платные. Последняя версия для Windows стоит около $49 долларов США за минимальную версию, и $89 за полную.

Настраиваются под индивидуальные предпочтения все компоненты: от тулбара до горячих клавиш. Есть как стандартный вид, так и двухпанельный (также есть трехпанельный). Поддерживается работа с FTP, встроенный zip-архиватор, свой процесс копирования, предпросмотр файлов, переключение видов в один клик, запоминание любимых папок и многое-многое другое

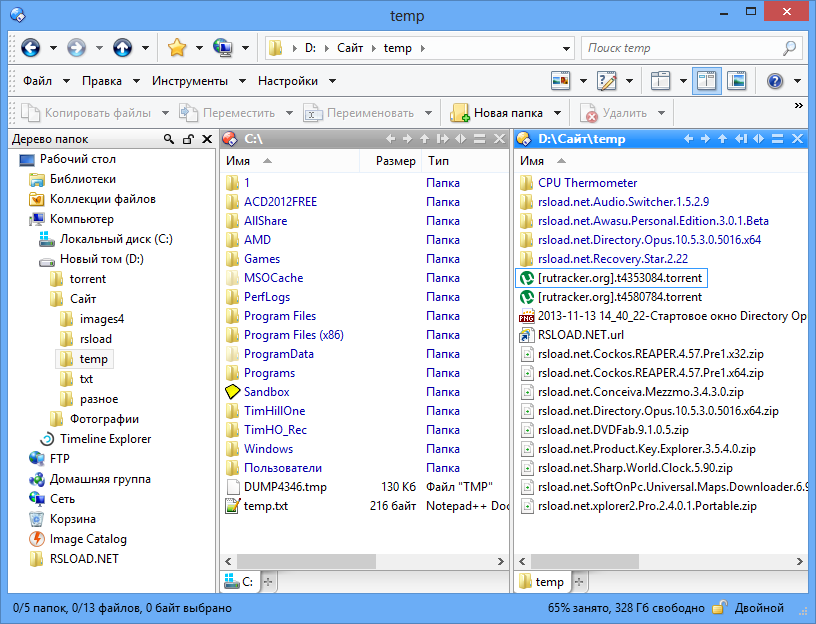


Рисунок 2.2 – Скриншот Directory Opus

### Windows Explorer

Проводник Windows — это приложение, реализующее графический интерфейс доступа пользователя к файлам в современных операционных системах Microsoft Windows и ReactOS. Проводник, в настоящее время, фактически является основой графической оболочки пользователя Windows. Проводник Windows впервые появился в Windows 95, как замена менеджера файлов Windows 3.x. Фактически, всё, что видит пользователь после загрузки, это Проводник Windows. Но иногда Проводником называют его часть, предназначенную для манипуляции файлами. Из интересных возможностей в Windows 8 Microsoft добавила нативную поддержку формата ISO, и теперь вы можете монтировать образы этого типа прямо в «Проводнике», выбрав соответствующий пункт контекстного меню, также есть возможность добавлять теги, комментарии и авторство к файлам.

Именно Проводник Windows был примером для подражания в аспектах визуально приятного и не перегруженного интерфейса, расположения элементов и интуитивности в выполнении базовых действий с файлами.

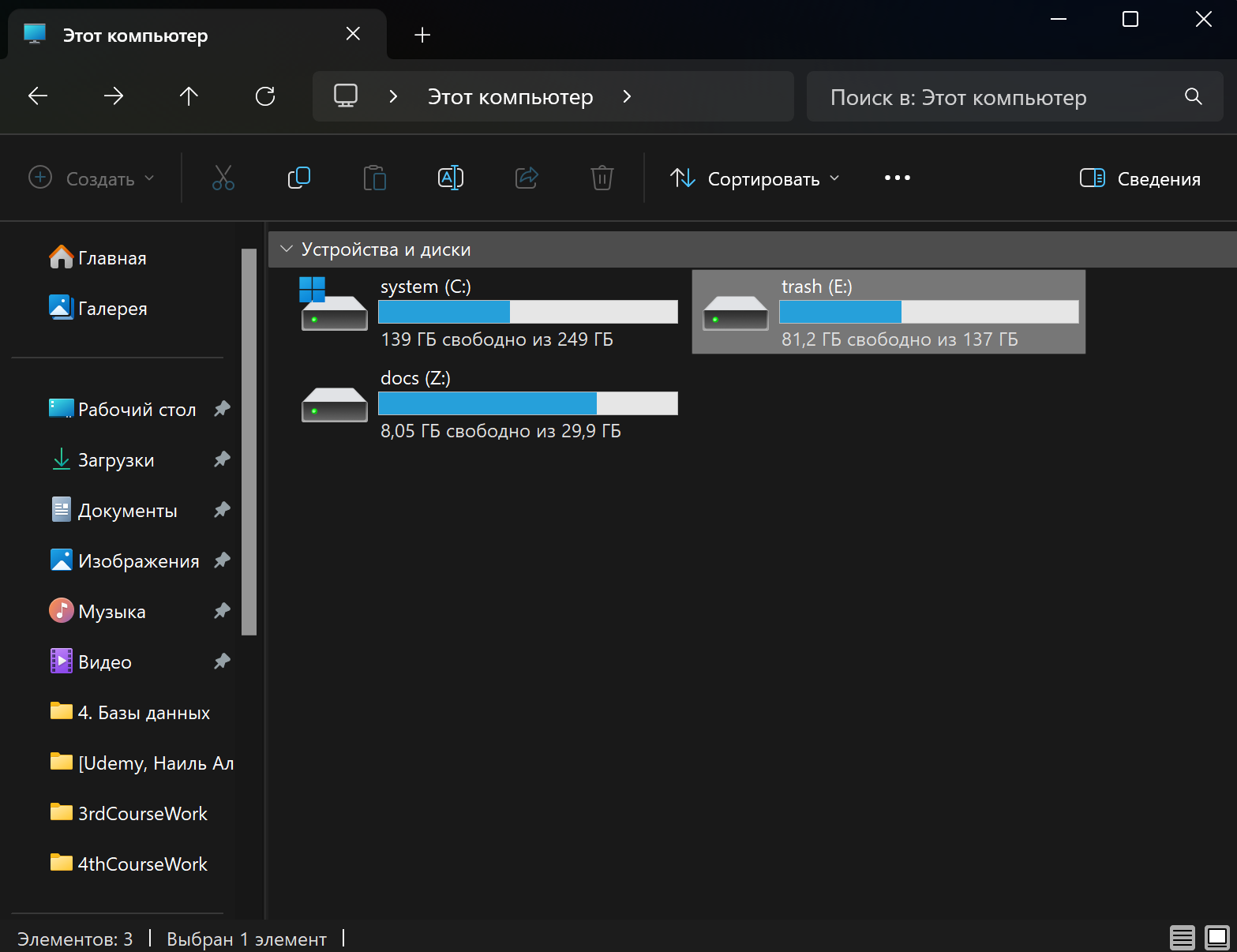


Рисунок 2.3 – Скриншот Windows Explorer

## Требования к работе программы

После рассмотрения аналогов данного курсового проекта, становится понятно, что подобного рода приложения, в основном, включают в себя основные функции:

* Множественные операции с файлами и папками, такие как копирование, вставление, удаление, перемещение.
* Навигация между различными директориями и файлами на компьютере.
* Просмотр содержимого папок в виде плиток или таблицы.
* Открывание и просмотр различных типов файлов, включая изображения, видео и документы.
* Копирование, перемещение и удаление файлов и папок.
* Редактирование метаданных файлов, таких как название и тип.
* Подключение и управление съемными носителями, такими как USB-флешки или внешние жесткие диски.
* Возможность ввода расположения файла или каталога в строку поиска.
* Для удобства навигации в файловой системе функции «назад» и «вперед».

Для создания приложения был выбран язык C++, так как он поддерживает использование объектно–ориентированной парадигмы, что делает его удобным для создания больших и сложных систем, разбитых на модули и классы. C++ остается одним из самых популярных и востребованных языков программирования. Полная информация об языке приведена в источниках [4] и [5].

Qt Creator, интегрированная среда разработки, была выбрана, так как предоставляет набор инструментов для написания, отладки и развертывания приложений, что упрощает процесс работы разработчиков. Сама же среда разработки содержит огромное множество недоработок, неудобств и проблем, однако в разработке на языке C++ является бесконкурентной по возможностям. Используется в Autodesk, Maya, Skype, Telegram, VirtualBox, Google, Panasonic, Philips, Samsung.

# СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

После определения требований к функционалу разрабатываемому приложению его следует разбить на функциональные блоки. Такой подход упростит понимание создания приложения, а также позволит легко оптимизацию.

## Модуль основного окна

Этот модуль представляет основное окно в приложении файлового менеджера для Windows, реализованного с использованием Qt. Он содержит множество функций для управления интерфейсом пользователя и взаимодействия с файловой системой. Его функционал включает:

* Кнопки для навигации «назад», «вперед» и «вверх» по файловой системе.
* Текстовое поле с отображением текущего пути и перехода по нему, а также с возможностью дописывать нужный адрес перехода.
* Кнопка для подтверждения введенного пути.
* Представление списка файлов и папок в виде иконок (mListView) и дерева (TreeView).
* Удаление, вставка, копирование файлов и папок.
* Определение текущего местоположения пользователя.
* Переключение между различными типами представления файлов и папок

## Модуль файловой системы

Этот модуль является подклассом QFileSystemModel в Qt, который представляет модель файловой системы для работы с файлами и папками на устройстве. Его функционал включает:

* Переключение между различными типами представления файлов и папок
* Поддержка перетаскивания и сбрасывания файлов и папок.
* Работа с MIME данными. MIME – это как язык, который помогает компьютерам понимать, как обрабатывать и показывать разные типы файлов, будь то тексты, картинки или музыка.
* Установка данных для элемента модели.
* Установка флагов, определяющих свойства для объектов модели, такие как редактируемость и возможность перетаскивания.
* Использование возможностей класса QFileSystemModel.

## Модуль отслеживания истории

Этот модуль представляет стек истории навигации в файловом менеджере, который позволяет отслеживать перемещение пользователя по файловой системе и предоставляет возможность переходить назад и вперед по истории навигации. Его функционал включает:

* Использование возможностей класса QFileSystemModel.
* Используется стек (QStack) для хранения путей к файлам и папкам.
* Отслеживание текущего индекса в стеке, а также для определения доступности операций «назад» и «вперед» в навигации.
* Возможность перемещаться вперед и назад по истории навигации с отображением файловой системы в соответствии с выбранной кнопкой.
* Добавления нового пути в историю навигации.
* Возвращение пути к файлу или папке, находящемуся впереди, назад или на текущем индексе в стеке.
* Определены сигналы, эмитируемые при изменении доступности кнопок «назад» и «вперед».

## Модуль меню при нажатии ПКМ

Этот модуль отвечает за меню по нажатии правой кнопки мыши. Его функционал включает:

* Создание контекстного меню по нажатии правой кнопки мыши.
* Совершение копирования файлов и каталогов.
* Совершение вставления файлов и каталогов.
* Совершение удаления файлов и каталогов.
* Совершение действия обновления отображения
* Возможность переключения вида отображения файловой системы. На выбор плиточный дизайн и дизайн в виде списка с возможностью сортировки содержимого каталога по различным папаметрам

# ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описывается функционирование и структура разрабатываемого приложения.

Диаграмма классов представлена в Приложении Б.

## Входные и выходные данные

Входными данными в приложении являются существующие разделы подключенных внешних ПЗУ. (Рисунок 4.1).

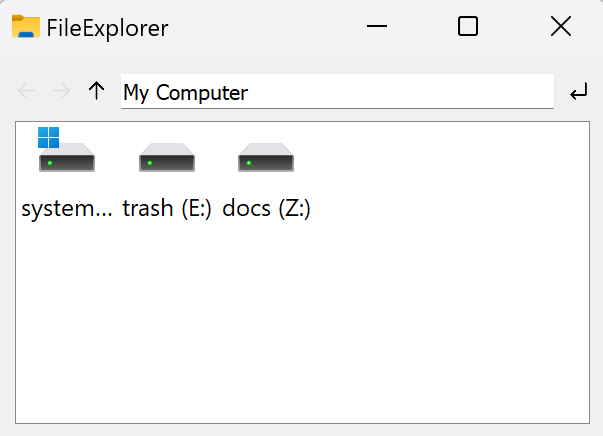


Рисунок 4.1 – Начальный экран

Также идет инициализация игрового процесса и интерфейса в MainWindow, которая включает огромное множество вызовов методов connect(), которые реализовывают поступающие сигналы извне. Также реализуются декларативным методом разработки все элементы интерфейса с прописыванием всех настроек вручную без реализации файла с расширением .ui. В этот список входят кнопки, панель текстового ввода, а также объекты Qt, такие как QListView и QTreeView.



Рисунок 4.2 – Инициализация начальных данных

В конце при открытии любого из дисков отображаются каталоги и файлы, которые действительно содержатся в системе. (Рисунок 4.3)

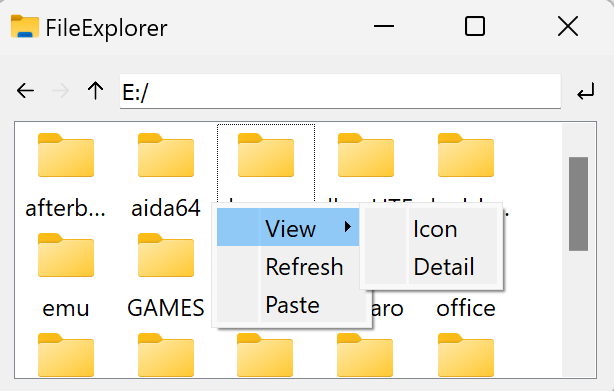


Рисунок 4.3 – Выходные данные с отображением папок и файлов

## Модуль основного окна

### Класс MainWindow

Этот класс является основным в приложении. Отвечает отображение графического интерфейса и получение указаний по его изменению от других классов. Поля:

* QPushButton\* mBackButtonView – указатель на кнопку навигации назад.
* QPushButton\* mForwardButtonView – указатель на кнопку навигации вперед.
* QPushButton\* mUpButtonView – указатель на кнопку навигации вверх.
* QPushButton\* mEnterButtonView – указатель на кнопку подтверждения введенного пути.
* QLineEdit\* mEditTextView – указатель на поле ввода, в котором отображается текущий путь.
* QListView\* mListView – указатель на виджет для отображения списка файлов и папок в виде списков.
* QTreeView\* mTreeView – указатель на виджет для отображения файловой системы в виде древовидной структуры.
* FileSystemModel\* mFileSystemModel – указатель на модель данных, используемая для представления файловой системы в виде списка или дерева.
* HistoryStack\* mHistoryStack – указатель на объект, представляющий стек истории навигации по файловой системе.
* QString mCurrPath – строка, содержащая текущий путь в файловой системе.
* ViewType mCurrType – определение типа представления файлов и папок.

Методы:

* MainWindow(QWidget \*parent) – конструктор, внутри которого проходит инициализация графического экрана.
* ~*MainWindow*() – освобождает ресурсы, связанные с объектом класса.
* int copySingleFile(*const* QString& src, *const* QString& dst, bool move) – копирует или перемещает файл.
* int copyDir(*const* QString& src, *const* QString& dst, bool move) – копирует или перемещает директорию.
* void initWidget() – инициализирует виджеты интерфейса и связывает их с соответствующими сигналами и слотами.
* bool isInMyComputer() *const* ‑ проверяет, находится ли текущий путь в «Моем компьютере».
* bool isInNetwork() *const –* проверяет, находится ли текущий путь в сетевом ресурсе.
* void jumpToMyComputer() – переходит к корневому каталогу «Мой компьютер».
* bool jumpTo(*const* QString& path) – переходит по указанному пути в файловой системе.
* QAbstractItemView \*getCurrentView() – возвращает указатель на текущее представление файлов и папок.
* void onLineEditEnter() – обрабатывает событие нажатия Enter в поле ввода пути.
* void onUpButtonClick() – обрабатывает событие нажатия кнопки "Вверх".
* void onBackButtonEnable(bool) – обрабатывает изменение доступности кнопки "Назад".
* void onForwardButtonEnable(bool) – обрабатывает изменение доступности кнопки "Вперед".
* void onBackButtonClick() – обрабатывает событие нажатия кнопки "Назад".
* void onForwardButtonClick() – обрабатывает событие нажатия кнопки "Вперед".
* void onItemDoubleClick(const QModelIndex& index)– обрабатывает событие двойного клика на элемент списка или дерева.
* void onCustomContextMenuRequested(const QPoint&) – обрабатывает запрос на контекстное меню.
* void onRefresh() – обновляет содержимое представления.
* void onRemove(const QString&) – обрабатывает запрос на удаление файла или папки.
* void onSwitchViewType(ViewType type) – обрабатывает запрос на изменение типа представления файлов и папок.
* void onPaste(const QStringList& srcs, const QString& dst, bool move) – обрабатывает запрос на вставку файлов или папок.

## Модуль файловой системы

### Класс FileSystemModel

Данный код представляет класс FileSystemModel, который является подклассом QFileSystemModel в Qt, что представляет модель файловой системы для работы с файлами и папками на устройстве

Методы:

* Qt::DropActions *supportedDropActions*() *const* *override ‑* возвращает действия, которые поддерживаются при перетаскивании элементов в модели, такие как копирование и перемещение.
* Qt::DropActions *supportedDragActions*() *const* *override –* возвращает действия, которые поддерживаются при перетаскивании элементов из модели, такие как копирование и перемещение.
* Qt::ItemFlags *flags*(*const* QModelIndex &index) *const* *override ‑* возвращает флаги для элементов модели, определяющие их свойства, такие как редактируемость и возможность перетаскивания.
* QStringList *mimeTypes*() *const* *override ‑* возвращает список поддерживаемых типов MIME данных модели. В данном случае, это те же типы, которые поддерживает базовая модель файловой системы QFileSystemModel.
* QMimeData \**mimeData*(*const* QModelIndexList& indexes) *const* *override ‑* возвращает MIME данные для выбранных элементов модели. В данном случае, просто передает запрос базовой модели QFileSystemModel и возвращает полученные MIME данные.
* bool *dropMimeData*(*const* QMimeData\* data, Qt::DropAction action, int row, int column, *const* QModelIndex& parent)–обрабатывает событие отпускания перетаскиваемых элементов на модель.
* bool *setData*(*const* QModelIndex& index, *const* QVariant &value, int role) *override* – делегирует запрос базовой модели.

## Модуль отслеживания истории

### Класс HistoryStack

Класс используется для представления стека истории навигации в файловом менеджере, который позволяет отслеживать перемещение пользователя по файловой системе и предоставляет возможность переходить назад и вперед по истории навигации.

Поля:

* QStack<QString> mPathStack ‑ стек, который содержит строки - пути, представляющие историю навигации.
* int mCurrIndex ‑ индекс текущего элемента в стеке mPathStack.
* bool mIsBackEnabled ‑ логическая переменная, которая определяет, доступна ли кнопка «назад» для использования.
* bool mIsForwardEnabled ‑ логическая переменная, которая определяет, доступна ли кнопка «вперед» для использования.

Методы:

* HistoryStack(QObject\* parent = *nullptr*) – конструктор класса.
* QString forwardPath() *const* – возвращает путь, на который можно переместиться вперед в истории навигации.
* QString backPath() *const* – возвращает путь, на который можно переместиться назад в истории навигации.
* QString currentPath() *const* – возвращает текущий путь в истории навигации.
* void forward() – используется для перемещения вперед по истории навигации.
* void back() – используется для перемещения назад по истории навигации.
* void push(*const* QString&) – используется для добавления нового пути в историю навигации.
* *inline* bool isEmpty() – проверяет, пуст ли стек истории навигации или текущий индекс равен -1.

## Модуль меню при нажатии ПКМ

### Класс RightClickMenu

Класс наследуется от класса Qt QMenu, и от него наследуются другие классы, работа которых заключается в реализации меню, создаваемого по нажатии правой кнопки мыши.

Методы:

* *explicit* RightClickMenu(QWidget \*parent = *nullptr*) – конструктор класса.

### Класс ItemMenu

Класс наследуется от класса RightClickMenu и представляет контекстное меню для элементов, в котором пользователь может выполнить действия над элементами, такими как вырезание, копирование и удаление.

Поля:

* QAction\* mCutAction ‑ пункт меню выполняет операцию вырезания выбранных элементов.
* QAction\* mCopyAction ‑ пункт меню выполняет операцию копирования выбранных элементов.
* QAction\* mRemoveAction ‑ пункт меню выполняет операцию удаления выбранных элементов.
* QStringList mSelectedItemList ‑ список выбранных элементов, над которыми будут выполняться действия.

Методы:

* *explicit* ItemMenu(QWidget \*parent = *nullptr*) – конструктор класса.
* *static ItemMenu\* instance()* – возвращает указатель на единственный экземпляр класса ItemMenu.
* *void setItemList(QStringList& list)* – устанавливает список выбранных элементов, над которыми будет выполняться действие контекстного меню.
* *void initWidget()* – инициализирует виджет контекстного меню и его действия.
* *void onCutAction()* – обрабатывает событие «вырезать».
* void onCopyAction() – обрабатывает событие скопировать».
* void onRemoveAction()– обрабатывает событие «удалить».

### Класс BlankMenu

Этот класс представляет собой контекстное меню, которое появляется при нажатии правой кнопки мыши.

Поля:

* QMenu\* mViewMenu ‑ меню для отображения дополнительных пунктов меню.
* QString mCurrPath ‑ текущий путь в проводнике, который используется для операций вставки и других операций, где требуется указать местоположение.
* QAction\* mViewIconAction ‑ действие меню для переключения отображения на иконки.
* QAction\* mViewDetailAction ‑ действие меню для переключения отображения на детали.
* QAction\* mPasteAction ‑ действие меню для вставки содержимого из буфера обмена.
* QAction\* mRefreshAction ‑ действие меню для обновления содержимого.

Методы:

* void initWidget() ‑ инициализирует виджет контекстного меню и его действия.
* static BlankMenu\* instance() ‑ возвращает указатель на единственный экземпляр класса BlankMenu.
* void setPath(const QString& info) ‑ устанавливает текущий путь в проводнике.
* void sigSwitchViewType(MainWindow::ViewType) ‑ Сигнал, отправляемый при выборе пункта меню Icon или Detail для изменения типа отображения.

# РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

В данном разделе рассмотрены пошаговые описания одних из важнейших алгоритмов, используемых в программе.

## Алгоритм копирования единичного файла

Для алгоритма по шагам рассмотрен метод int MainWindow::copySingleFile(*const* QString &src, *const* QString &dst, bool move) класса MainWindow.

Этот метод предназначен для копирования или перемещения одиночного файла из одного места в другое, с возможностью замены существующего файла в пункте назначения и обработки ошибок в процессе. Возможной ошибкой, к примеру, является копирование файла из одной директории в директорию, где файл с таким названием уже существует. В таком случае пользователю дается выбор или скопировать новый файл, или оставить старый.

Шаг 1. Начало алгоритма copySingleFile.

Шаг 2. Получить информации о файле src при помощи QFileInfo srcFileinfo(src) и проверка существования файла и его типа. Если файл не существует или является директорией, выдается сообщение об ошибке и возвращается COPY\_SKIP.

Шаг 3. Сравнить пути src и dst. Если они совпадают, вернуть COPY\_OK.

Шаг 4. Получить информации о файле dst и создать объект QFile dstFile(dst). Проверить существования файла dst. Если файл существует, то в зависимости от выбора пользователя либо файл заменить, либо копирование пропустить.

Шаг 5. Если копирование произошло успешно, и если указано перемещение, то исходный файл удалить.

Шаг 6. Вернуть статус копирования (COPY\_OK, COPY\_SKIP или COPY\_ERROR).

Шаг 7. Конец алгоритма.

Схема алгоритма представлена в Приложении Б.

## Алгоритм обработки отпускания перетаскиваемых элементов

Для алгоритма по шагам рассмотрен метод bool FileSystemModel::*м*(*const* QMimeData \*data, Qt::DropAction action, int row, int column, *const* QModelIndex &parent) класса FileSystemModel.

Этот метод обрабатывает событие отпускания перетаскиваемых элементов на модель. Он извлекает локальные пути к файлам из переданных данных и генерирует сигнал, содержащий эти пути и информацию о родительской директории и действии (перемещение или копировани

Шаг 1. Начало алгоритма.

Шаг 2. Проверить, было ли действие action игнорировано (Qt::IgnoreAction). Если да, вернуть false, игнорируя событие.

Шаг 3. Проверить наличие URL в переданных данным data. Если URL отсутствуют, вернуть false.

Шаг 4. Получить список URL из переданных данных data.

Шаг 5. Создать список строк pathList, в котором будут храниться локальные пути к файлам.

Шаг 6. Итерация по списку URL. Если URL не является локальным файлом, перейти к следующему URL.

Шаг 7. Добавить локального пути файла в список pathList.

Шаг 8. Проверить, пуст ли список pathList. Если пуст, вернуть false.

Шаг 9. Генерировать сигнал sigDropEnd, который отправляет список локальных путей файлов, путь родительской директории parent, а также информацию о том, было ли выполнено действие перемещения (перетаскивание).

Шаг 10. Вернуть true в случае успешной обработки события.

Шаг 11. Конец алгоритма.

Схема алгоритма представлена в Приложении В.

# ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

В данном разделе описывается функциональное тестирование программы.

## Тестирование ввода некорректного пути в строку ввода

Приложение обрабатывает ситуацию ввода несуществующего пути, выдавая ошибку (Рисунок 6.1).

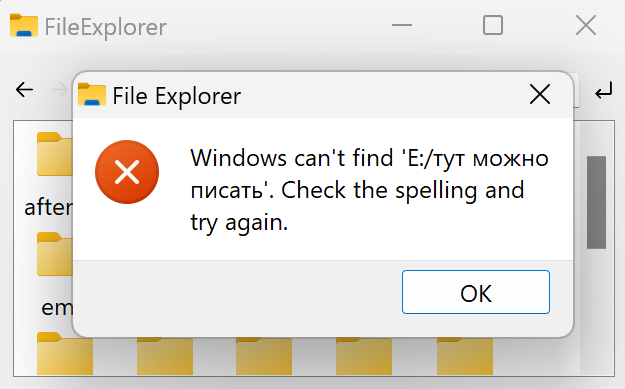


Рисунок 6.1 – Вывод ошибки о несуществующем пути

## Тестирование копирование файла в его же каталог

Приложение не позволит копировать файл с тем же названием в тот же каталог (Рисунок 6.2).

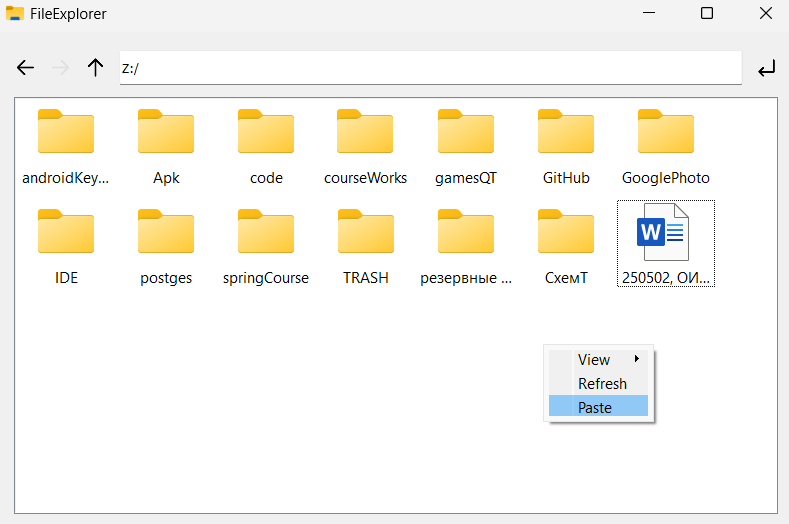


Рисунок 6.2 – Отображение попытки копирования файл в тот же каталог

## Тестирование попытки переименования файла в уже существующее название

При попытке переназвать файл в уже занятое название в этом каталоге выскакивает предупреждение, после чего название файла остается прежним (Рисунок 6.3).

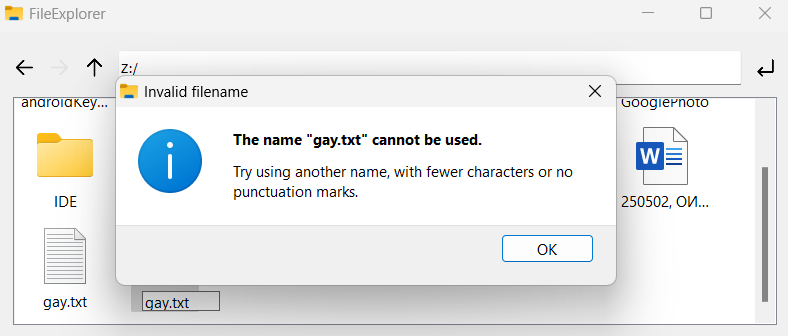


Рисунок 6.3 – Предупреждение о невозможности переименования файла

## Тестирование попытки переноса файла с идентичным названием

При попытке вставить файл из другого каталога в каталог, в котором уже есть файл с таким названием, выскакивает AlertDialog с возможностью выбора дальнейших действий(Рисунок 6.4).

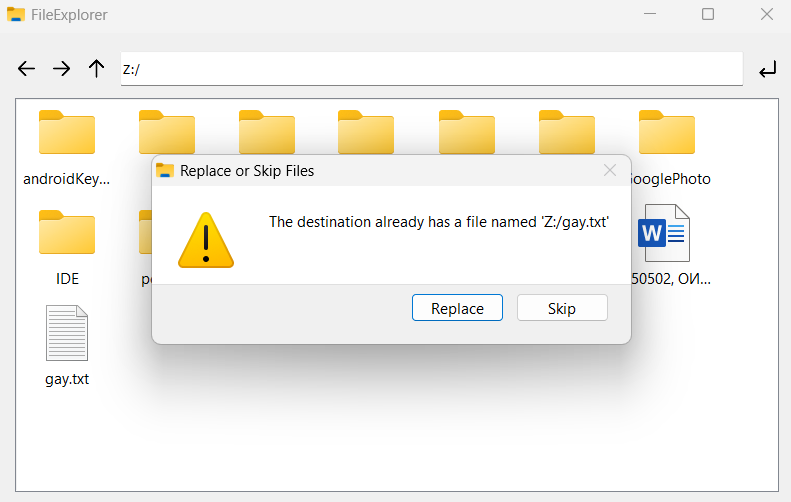


Рисунок 6.4 – Вывод возможного выбора при переносе файлов с одинаковыми названиями

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## Системные требования

Данное приложение разработано в QT Creator на 64–разрядной операционной системе Windows 11. Процессор Intel Core i7–12700h, размер оперативной памяти 16 Гб. Для нормальной работы данного приложения требуется не менее 20 Мб свободной оперативной памяти.

## Использование приложения

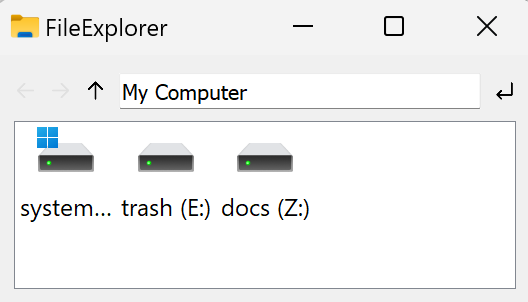
Для запуска программы необходим открыть файлы исходного кода в Qt Creator и собрать проект. Информация по созданию проекта находится в источнике [6]. Когда программа запустится, она готова к использованию (Рисунок 7.1).

Рисунок 7.1 – Главный экран

После этого отобразится окно со всеми разделами всех дисков и flash-накопителей. При двойном нажатии на любой раздел диска происходит переход по указанному пути, где отображаются все каталоги с возможностью перехода в них и файлы с возможностью их открытия, удаления, копирования и вставки (Рисунок 7.2).

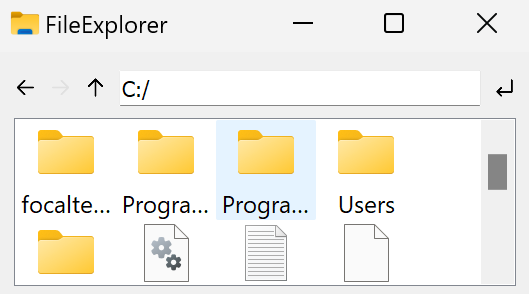


Рисунок 7.2 – Главное меню

При нажатии правой кнопкой мыши по пустому простанству выскакивает контекстное меню с возможностью выбора различных опций (Рисунок 7.3).

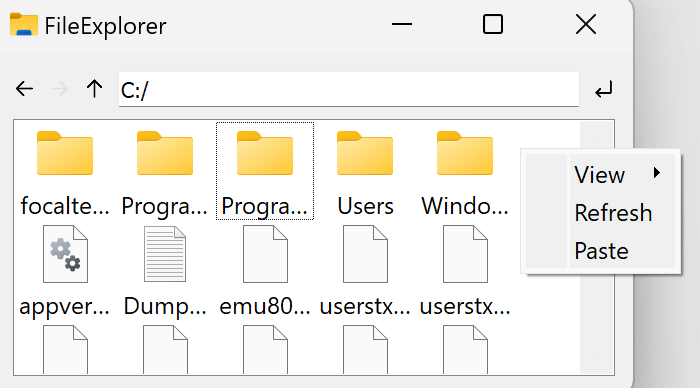


Рисунок 7.3 – Отображение контекстного меню при нажатии ПКМ на свободную область

При нажатии правой кнопкой мыши по файлу или каталогу выскакивает контекстное меню с возможностью выбора различных опций (Рисунок 7.4)

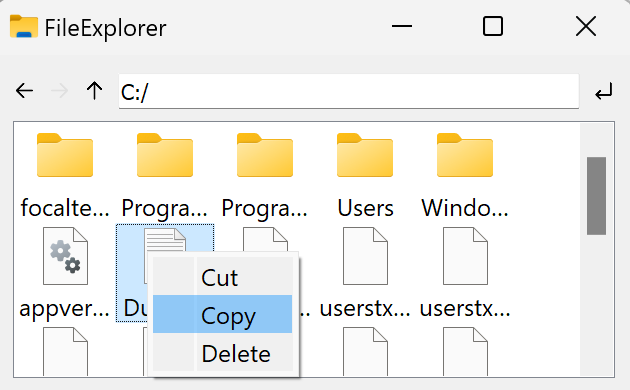


Рисунок 7.4 – Отображение контекстного меню при нажатии ПКМ на файл или каталог

При выделении группы файлов и нажатии ПКМ появляется контекстное меню (Рисунок 7.5).

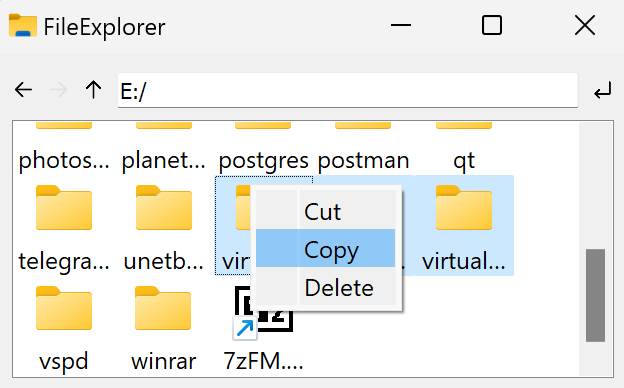


Рисунок 7.5 – Контекстное меню для группы файлов

При вводе в строку для ввода некоректного пути выскакивает ошибка

(Рисунок 7.6).

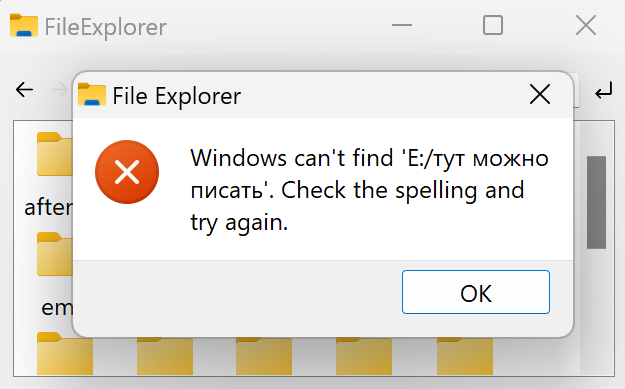


Рисунок 7.6 – Ошибка при вводе неверного пути

ЗАКЛЮЧЕНИЕ

Данный курсовой проект представляет собой разработанное приложение, которое обладает широким функционалом и интуитивно понятным пользовательским интерфейсом. В ходе его создания были успешно достигнуты поставленные цели, и весь запланированный функционал был реализован полностью и эффективно.

Для создания программного продукта была проведена детальная изучение среды разработки Qt Creator. Исследования помогли получить глубокое понимание особенностей каждого элемента, что сыграло ключевую роль в успешной реализации проекта.

Основное внимание уделено языку программирования C++, где были усвоены основы объектно-ориентированного программирования (ООП), что позволило эффективно использовать его возможности при создании различных модулей приложения.

Работа над проектом была разбита на этапы: анализ аналогов и литературных источников, постановка требований, проектирование, конструирование, разработка модулей и тестирование. Благодаря последовательности выполнения каждого этапа был достигнут результат – рабочая версия файлового менеджера с расширенным функционалом для Windows, который не стал полной заменой для встроенного файлового менеджера, однако является его легковесной упрощенной копией.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] C++ GUI Programming with Qt 4, – Jasmin Blanchette, Mark Summerfield, 2015

[2] Microsoft Learn [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com – Дата доступа: 06.04.2024

[3] Файловые менеджеры для Windows [Электронный ресурс]. – Режим доступа: https://www.movavi.ru/learning-portal/best-file-manager.html – Дата доступа: 05.04.2024

[4] The C++ Programming Language, – Bjarne Stroustrup, 1985

[5] The C Programming Language. 2nd Edition, – Dennis Ritchie, Brian Kernighan, 1978

[6] QT Documentation [Электронный ресурс]. – Режим доступа: <https://doc.qt.io/> – Дата доступа: 05.04.2024

[7] Web-сервис «GitHub» [Электронный ресурс]. – bsuirSchedule – Режим доступа: <https://github.com/andrejHurynovic/bsuirSchedule> – Дата доступа: 05.04.2024

[8] Интегрированная информационная система «БГУИР: Университет» [Электронный ресурс]. – Документация – Режим доступа: https://iis.bsuir.by/api – Дата доступа: 06.04.2024

ПРИЛОЖЕНИЕ А

(обязательное)

**Диаграмма классов**

ПРИЛОЖЕНИЕ Б

(обязательное)

**Схема алгоритма копирования единичного файла**

ПРИЛОЖЕНИЕ В

(обязательное)

**Схема алгоритма обработки отпускания перетаскиваемых элементов**

ПРИЛОЖЕНИЕ Г

(обязательное)

**Листинг кода**

#ifndef FILESYSTEMMODEL\_H

#define FILESYSTEMMODEL\_H

#include <QFileSystemModel>

class FileSystemModel : public QFileSystemModel

{

Q\_OBJECT

public:

explicit FileSystemModel(QObject \*parent = nullptr) {};

public:

Qt::DropActions supportedDropActions() const override;

Qt::DropActions supportedDragActions() const override;

Qt::ItemFlags flags(const QModelIndex &index) const override;

QStringList mimeTypes() const override;

QMimeData \*mimeData(const QModelIndexList& indexes) const override;

bool dropMimeData(const QMimeData\* data, Qt::DropAction action, int row, int column, const QModelIndex& parent) override;

bool setData(const QModelIndex& index, const QVariant &value, int role) override;

signals:

void sigDropEnd(const QStringList&, const QString&, bool);

};

#endif // FILESYSTEMMODEL\_H

#ifndef HISTORYSTACK\_H

#define HISTORYSTACK\_H

#include <QStack>

#include <QObject>

class HistoryStack : public QObject {

Q\_OBJECT

public:

HistoryStack(QObject\* parent = nullptr);

private:

QStack<QString> mPathStack;

int mCurrIndex;

bool mIsBackEnabled;

bool mIsForwardEnabled;

public:

QString forwardPath() const;

QString backPath() const;

QString currentPath() const;

void forward();

void back();

void push(const QString&);

inline bool isEmpty() {return mPathStack.isEmpty() || mCurrIndex == -1;}

signals:

void sigBackBtnIsEnable(bool);

void sigForwardBtnIsEnable(bool);

};

#endif // HISTORYSTACK\_H

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#define COPY\_OK 0

#define COPY\_SKIP -1 // Пользователь пропускает

#define COPY\_ERROR -2 // Ошибка API

#include <QMainWindow>

class QPushButton;

class QLineEdit;

class QListView;

class QTreeView;

class FileSystemModel;

class QAbstractItemView;

class HistoryStack;

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

enum ViewType{Icon, Detail};

Q\_ENUM(ViewType)

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private:

QPushButton\* mBackButtonView;

QPushButton\* mForwardButtonView;

QPushButton\* mUpButtonView;

QLineEdit\* mEditTextView;

QPushButton\* mEnterButtonView;

QListView\* mListView;

QTreeView\* mTreeView;

FileSystemModel\* mFileSystemModel;

HistoryStack\* mHistoryStack;

QString mCurrPath;

ViewType mCurrType;

public:

///Foucs: dst is destination file path (include 1.txt)

int copySingleFile(const QString& src, const QString& dst, bool move);

///Focus: dst is destination dir path (targetDir/)

int copyDir(const QString& src, const QString& dst, bool move);

private:

void initWidget();

bool isInMyComputer() const;

bool isInNetwork() const;

void jumpToMyComputer();

bool jumpTo(const QString& path);

QAbstractItemView \*getCurrentView();

private slots:

void onLineEditEnter();

void onUpButtonClick();

void onBackButtonEnable(bool);

void onForwardButtonEnable(bool);

void onBackButtonClick();

void onForwardButtonClick();

void onItemDoubleClick(const QModelIndex& index);

void onCustomContextMenuRequested(const QPoint&);

void onRefresh();

void onRemove(const QString&);

void onSwitchViewType(ViewType type);

void onPaste(const QStringList& srcs, const QString& dst, bool move);

};

#endif // MAINWINDOW\_H

#ifndef RIGHTCLICKMENU\_H

#define RIGHTCLICKMENU\_H

#define BLANKMENUINSTANCE BlankMenu::instance()

#include <QMenu>

#include "MainWindow.h"

class RightClickMenu : public QMenu {

Q\_OBJECT

public:

explicit RightClickMenu(QWidget \*parent = nullptr);

};

class ItemMenu : public RightClickMenu {

Q\_OBJECT

private:

explicit ItemMenu(QWidget \*parent = nullptr);

private:

QAction\* mCutAction;

QAction\* mCopyAction;

QAction\* mRemoveAction;

QStringList mSelectedItemList;

public:

static ItemMenu\* instance();

void setItemList(const QStringList& list);

private:

void initWidget();

signals:

void sigRemove(const QString&);

private slots:

void onCutAction();

void onCopyAction();

void onRemoveAction();

};

#define ITEMMENUINSTANCE ItemMenu::instance()

class BlankMenu : public RightClickMenu {

Q\_OBJECT

private:

explicit BlankMenu(QWidget \*parent = nullptr);

private:

QMenu\* mViewMenu;

QString mCurrPath;

QAction\* mViewIconAction;

QAction\* mViewDetailAction;

QAction\* mPasteAction;

QAction\* mRefreshAction;

private:

void initWidget();

public:

static BlankMenu\* instance();

void setPath(const QString& info);

private:

signals:

void sigSwitchViewType(MainWindow::ViewType);

void sigRefresh();

void sigPaste(const QStringList&, const QString&, bool);

private slots:

void onViewIconAction();

void onViewDetailAction();

void onPasteAction();

};

#endif // RIGHTCLICKMENU\_H

#include "FileSystemModel.h"

#include <QMimeData>

#include <QUrl>

//Возвращает действия, поддерживаемые при перетаскивании элементов в модели (копирование, перемещение и т. д.)

Qt::DropActions FileSystemModel::supportedDropActions() const {

return Qt::CopyAction | Qt::MoveAction;

}

//Возвращает действия, поддерживаемые при перетаскивании элементов из модели (копирование, перемещение и т. д.)

Qt::DropActions FileSystemModel::supportedDragActions() const {

return Qt::CopyAction | Qt::MoveAction;

}

//Возвращает флаги для элементов модели, определяющие их свойства (редактируемость, возможность перетаскивания и т. д.)

Qt::ItemFlags FileSystemModel::flags(const QModelIndex &index) const {

Qt::ItemFlags flag = QFileSystemModel::flags(index);

if(index.isValid()) {

flag = flag | Qt::ItemIsEditable;

if(fileInfo(index).isDir()) {

flag = flag | Qt::ItemIsDropEnabled;

}

}

//qDebug() << flag << endl;

return flag;

}

// Возвращает список поддерживаемых типов MIME данных модели

QStringList FileSystemModel::mimeTypes() const {

return QFileSystemModel::mimeTypes();

}

//Возвращает MIME данные для выбранных элементов модели

QMimeData \*FileSystemModel::mimeData(const QModelIndexList &indexes) const {

QMimeData\* ori = QFileSystemModel::mimeData(indexes);

return ori;

}

//Обрабатывает событие отпускания перетаскиваемых элементов на модель

bool FileSystemModel::dropMimeData(const QMimeData \*data,

Qt::DropAction action,

int row, int column,

const QModelIndex &parent) {

if(action == Qt::IgnoreAction) {

return false;

}

if(!data->hasUrls()) {

return false;

}

QList<QUrl> urls = data->urls();

QStringList pathList;

for(QUrl url : urls) {

if(!url.isLocalFile()) {

continue;

}

pathList << url.toLocalFile();

}

if(pathList.empty()) {

return false;

}

emit sigDropEnd(pathList, fileInfo(parent).absoluteFilePath(), action == Qt::MoveAction);

return true;

}

//Устанавливает данные для элемента модели

bool FileSystemModel::setData(const QModelIndex &index, const QVariant &value, int role) {

return QFileSystemModel::setData(index, value, role);

}

#include "HistoryStack.h"

HistoryStack::HistoryStack(QObject\* parent) : QObject (parent), mCurrIndex(-1), mIsBackEnabled(false), mIsForwardEnabled(false) {};

QString HistoryStack::forwardPath() const {

return mCurrIndex < mPathStack.size() - 1 && mCurrIndex >= -1 ? mPathStack[mCurrIndex + 1] : QString();

}

QString HistoryStack::backPath() const {

return mCurrIndex > 0 && mCurrIndex <= mPathStack.size() ? mPathStack[mCurrIndex - 1] : QString();

}

QString HistoryStack::currentPath() const {

return mCurrIndex >= 0 && mCurrIndex < mPathStack.size() ? mPathStack[mCurrIndex] : QString();

}

//используется для перемещения вперед по навигации

void HistoryStack::forward() {

if(mCurrIndex < mPathStack.size() - 1) {

mCurrIndex++;

}

if(mCurrIndex > 0 && !mIsBackEnabled){

mIsBackEnabled = true;

emit sigBackBtnIsEnable(true);

}

if(mCurrIndex >= mPathStack.size() - 1) {

mIsForwardEnabled = false;

emit sigForwardBtnIsEnable(false);

}

}

//используется для перемещения назад по навигации

void HistoryStack::back() {

if(mCurrIndex > 0) {

mCurrIndex--;

}

if(mCurrIndex < mPathStack.size() - 1 && !mIsForwardEnabled) {

mIsForwardEnabled = true;

emit sigForwardBtnIsEnable(true);

}

if(mCurrIndex <= 0) {

mIsBackEnabled = false;

emit sigBackBtnIsEnable(false);

}

}

//используется для добавления нового пути в историю навигации

void HistoryStack::push(const QString& val) {

while(mPathStack.size() - 1 > mCurrIndex) {

mPathStack.pop();

}

mPathStack.push(val);

forward();

}

#include "MainWindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

QApplication::setWindowIcon(QIcon(":/icon/Resources/icon.png"));

MainWindow w;

w.show();

return a.exec();

}

#include "MainWindow.h"

#include "FileSystemModel.h"

#include "HistoryStack.h"

#include "RightClickMenu.h"

#include <QPushButton>

#include <QLineEdit>

#include <QListView>

#include <QTreeView>

#include <QHBoxLayout>

#include <QVBoxLayout>

#include <QMessageBox>

#include <Windows.h>

#include <QDesktopServices>

#include <QUrl>

#include <vector>

const QString FILE\_DIR\_NOT\_EXIST\_MSG = QObject::tr("Could not find this item.\n'%1'");

const QString ENTER\_DIR\_NOT\_EXIST\_MSG = QObject::tr("Windows can't find '%1'. Check the spelling and try again.");

const QString FILE\_EXIST\_MSG = QObject::tr("The destination already has a file named '%1'");

const QString FILE\_SAME\_WITH\_DIR\_MSG = QObject::tr("There is already a folder with the same name as the file name you specified.\n'%1'\nSpecifiy a different name");

const QString DIR\_SUB\_DIR\_MSG = QObject::tr("The destination folder is a subfolder of the source folder.\n'%1'");

const QString FILE\_REMOVE\_MSG = QObject::tr("Could not remove this item.\n'%1'");

//инициализирует виджеты, соединяет сигналы и слоты, а также настраивает начальное состояние

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

{

initWidget();

connect(mEnterButtonView, &QPushButton::clicked,this, &MainWindow::onLineEditEnter);

connect(mEditTextView, &QLineEdit::returnPressed,this, &MainWindow::onLineEditEnter);

connect(mListView, &QListView::doubleClicked,this, &MainWindow::onItemDoubleClick);

connect(mTreeView, &QTreeView::doubleClicked,this, &MainWindow::onItemDoubleClick);

connect(mUpButtonView, &QPushButton::clicked,this, &MainWindow::onUpButtonClick);

connect(mHistoryStack, &HistoryStack::sigBackBtnIsEnable,this, &MainWindow::onBackButtonEnable);

connect(mHistoryStack, &HistoryStack::sigForwardBtnIsEnable,this, &MainWindow::onForwardButtonEnable);

connect(mBackButtonView, &QPushButton::clicked,this, &MainWindow::onBackButtonClick);

connect(mForwardButtonView, &QPushButton::clicked,this, &MainWindow::onForwardButtonClick);

//context menu

connect(mListView, &QWidget::customContextMenuRequested,this, &MainWindow::onCustomContextMenuRequested);

connect(mTreeView, &QWidget::customContextMenuRequested,this, &MainWindow::onCustomContextMenuRequested);

connect(BLANKMENUINSTANCE, &BlankMenu::sigSwitchViewType,this, &MainWindow::onSwitchViewType);

connect(BLANKMENUINSTANCE, &BlankMenu::sigRefresh,this, &MainWindow::onRefresh);

connect(BLANKMENUINSTANCE, &BlankMenu::sigPaste,this, &MainWindow::onPaste);

connect(ITEMMENUINSTANCE, &ItemMenu::sigRemove,this, &MainWindow::onRemove);

//drop

connect(mFileSystemModel, &FileSystemModel::sigDropEnd,this, &MainWindow::onPaste);

onSwitchViewType(Icon);

jumpToMyComputer();

mHistoryStack->push(mFileSystemModel->myComputer().toString());

}

MainWindow::~MainWindow() {

}

//Метод для копирования единичного файла из одного пути src в другой dst. Параметр move указывает, следует ли переместить файл (true) или скопировать его (false)

int MainWindow::copySingleFile(const QString &src, const QString &dst, bool move)

{

QFileInfo srcFileinfo(src);

if(!srcFileinfo.exists() || srcFileinfo.isDir()){

QMessageBox::critical(this, "Item Not Found", FILE\_DIR\_NOT\_EXIST\_MSG.arg(srcFileinfo.absoluteFilePath()), "Skip");

return COPY\_SKIP;

}

if(src.compare(dst) == 0) {

return COPY\_OK;

}

QFileInfo dstFileInfo(dst);

QFile dstFile(dst);

if(dstFileInfo.exists()) {

if(dstFileInfo.isFile()) {

if(QMessageBox::warning(this, "Replace or Skip Files", FILE\_EXIST\_MSG.arg(dstFileInfo.absoluteFilePath()),

"Replace", "Skip") == 0) {

if(!dstFile.remove()) {

return COPY\_ERROR;

}

} else {

return COPY\_SKIP;

}

} else {

QMessageBox::critical(this, "Rename File", FILE\_SAME\_WITH\_DIR\_MSG.arg(dstFileInfo.absoluteFilePath()), "Skip");

return COPY\_SKIP;

}

}

if(!QFile::copy(src, dst)) {

return COPY\_ERROR;

}

if(move) {

QFile::remove(src);

}

return COPY\_OK;

}

//Метод для копирования директории из одного пути src в другой dst. Параметр move указывает, следует ли переместить директорию (true) или скопировать ее (false)

int MainWindow::copyDir(const QString &src, const QString &dst, bool move)

{

QFileInfo srcFileinfo(src);

if(!srcFileinfo.exists() || srcFileinfo.isFile()){

QMessageBox::critical(this, "Item Not Found", FILE\_DIR\_NOT\_EXIST\_MSG.arg(srcFileinfo.absoluteFilePath()), "Skip");

return COPY\_SKIP;

}

if(src.compare(dst) == 0) return COPY\_OK;

QDir srcDir(src), dstDir(dst);

while(dstDir.cdUp()){

if(dstDir == srcDir){

QMessageBox::critical(this, "Subfolder", DIR\_SUB\_DIR\_MSG.arg(dstDir.absolutePath()), "Skip");

return COPY\_SKIP;

}

}

dstDir.setPath(dst);

QFileInfo srcFileInfo(src);

QFileInfo dstFileInfo(dst);

if(srcFileInfo.isFile()) {

return COPY\_ERROR;

}

if(dstFileInfo.exists() && dstFileInfo.isFile()) {

QMessageBox::critical(this, "Rename File", FILE\_SAME\_WITH\_DIR\_MSG.arg(dstFileInfo.absoluteFilePath()), "Skip");

return COPY\_SKIP;

}

if(!dstDir.exists()) {

if(!dstDir.mkdir(dstDir.absolutePath())) {

return COPY\_ERROR;

}

}

QFileInfoList srcFileInfoList = srcDir.entryInfoList();

for(QFileInfo srcFileInfo : srcFileInfoList) {

if(srcFileInfo.fileName() == "." || srcFileInfo.fileName() == "..") {

continue;

}

if(srcFileInfo.isDir()) {

if(copyDir(srcFileInfo.filePath(), dstDir.filePath(srcFileInfo.fileName()), move) == COPY\_ERROR) {

return COPY\_ERROR;

}

} else {

QFileInfo dstFileInfo(dstDir, srcFileInfo.fileName());

if(copySingleFile(srcFileInfo.absoluteFilePath(), dstFileInfo.absoluteFilePath(), move) == COPY\_ERROR) {

return COPY\_ERROR;

}

}

}

if(move && srcDir.isEmpty()) {

srcDir.rmpath(srcDir.absolutePath());

}

return COPY\_OK;

}

//Слот, вызываемый при изменении типа представления (иконки или строки)

void MainWindow::onSwitchViewType(MainWindow::ViewType type)

{

if(mCurrType != type) {

switch (type) {

case Icon:

mTreeView->hide();

mListView->setRootIndex(mFileSystemModel->index(mCurrPath));

mListView->show();

break;

case Detail:

mListView->hide();

mTreeView->setRootIndex(mFileSystemModel->index(mCurrPath));

mTreeView->show();

break;

}

mCurrType = type;

}

}

//Этот метод выполняет вставку скопированных или перемещенных файлов или директорий из списка srcs в папку с путем dst. Он перебирает каждый элемент в списке srcs, определяет, является ли он файлом или директорией, а затем использует методы CopySingleFile или CopyDir для копирования или перемещения соответственно

void MainWindow::onPaste(const QStringList &srcs, const QString &dst, bool move)

{

for(QString src : srcs) {

QFileInfo srcFileInfo(src);

QFileInfo dstFileInfo(QDir(dst), srcFileInfo.fileName());

if(srcFileInfo.isFile()) {

copySingleFile(src, dstFileInfo.absoluteFilePath(), move);

} else {

copyDir(src, dstFileInfo.absoluteFilePath(), move);

}

}

if(isInNetwork()) {

onRefresh();

}

}

//Этот метод создает и настраивает различные виджеты, такие как кнопки, поле ввода и просмотрщики файловой системы. Он устанавливает связи между сигналами и слотами для обработки действий пользователя

void MainWindow::initWidget() {

//layout

mBackButtonView = new QPushButton(this);

mForwardButtonView = new QPushButton(this);

mUpButtonView = new QPushButton(this);

mEditTextView = new QLineEdit(this);

mEnterButtonView = new QPushButton(this);

mListView = new QListView(this);

mTreeView = new QTreeView(this);

QHBoxLayout\* topLayout = new QHBoxLayout(this);

topLayout->addWidget(mBackButtonView);

topLayout->addWidget(mForwardButtonView);

topLayout->addWidget(mUpButtonView);

topLayout->addWidget(mEditTextView);

topLayout->addWidget(mEnterButtonView);

QVBoxLayout\* layout = new QVBoxLayout(this);

layout->addLayout(topLayout);

layout->addWidget(mListView);

layout->addWidget(mTreeView);

QWidget\* centralWidget = new QWidget(this);

centralWidget->setLayout(layout);

setCentralWidget(centralWidget);

//property

mFileSystemModel = new FileSystemModel();

mHistoryStack = new HistoryStack(this);

//history

mBackButtonView->setEnabled(false);

mForwardButtonView->setEnabled(false);

//view

mListView->setModel(mFileSystemModel);

mListView->setViewMode(QListView::IconMode);

mListView->setIconSize(QSize(80,80));

mListView->setGridSize(QSize(100,100));

mListView->setResizeMode(QListView::Adjust);

mListView->setMovement(QListView::Snap);

mListView->setContextMenuPolicy(Qt::CustomContextMenu);

mListView->setAcceptDrops(true);

mListView->setDragEnabled(true);

mListView->setDropIndicatorShown(true);

mListView->setDragDropMode(QAbstractItemView::DragDrop);

mListView->setSelectionMode(QAbstractItemView::ExtendedSelection);

mListView->setEditTriggers(QAbstractItemView::SelectedClicked);

mTreeView->setModel(mFileSystemModel);

mTreeView->setContextMenuPolicy(Qt::CustomContextMenu);

mTreeView->setRootIsDecorated(false);

mTreeView->setItemsExpandable(false);

mTreeView->setAcceptDrops(true);

mTreeView->setDragEnabled(true);

mTreeView->setDropIndicatorShown(true);

mTreeView->setDragDropMode(QAbstractItemView::DragDrop);

mTreeView->setSelectionMode(QAbstractItemView::ExtendedSelection);

mTreeView->setEditTriggers(QAbstractItemView::SelectedClicked);

mTreeView->setSortingEnabled(true);

mTreeView->sortByColumn(0, Qt::AscendingOrder);

//UI

mBackButtonView->setStyleSheet("QPushButton{border-image: url(:/icon/Resources/back-enable.png)}"

"QPushButton:disabled{border-image: url(:/icon/Resources/back-disable.png)}");

mForwardButtonView->setStyleSheet("QPushButton{border-image: url(:/icon/Resources/forward-enable.png)}"

"QPushButton:disabled{border-image: url(:/icon/Resources/forward-disable.png)}");

mUpButtonView->setStyleSheet("QPushButton{border-image: url(:/icon/Resources/up-enable.png)}"

"QPushButton:disabled{border-image: url(:/icon/Resources/up-disable.png)}");

mEnterButtonView->setStyleSheet("QPushButton{border-image: url(:/icon/Resources/enter.png)}");

mBackButtonView->setFixedSize(QSize(23,23));

mForwardButtonView->setFixedSize(QSize(23,23));

mUpButtonView->setFixedSize(QSize(23,23));

mEnterButtonView->setFixedSize(QSize(23,23));

resize(800, 500);

}

// проверяет, находится ли текущий путь в разделе "Мой компьютер"

bool MainWindow::isInMyComputer() const {

return mCurrPath.compare(mFileSystemModel->myComputer().toString()) == 0;

}

//проверяет, находится ли текущий путь в сетевой папке

bool MainWindow::isInNetwork() const {

return mCurrPath.startsWith("//") | mCurrPath.startsWith("\\\\");

}

//переключает текущий путь на раздел "Мой компьютер" и обновляет представление файловой системы

void MainWindow::jumpToMyComputer() {

mCurrPath = mFileSystemModel->myComputer().toString();

mEditTextView->setText(mCurrPath);

QModelIndex rootIndex = mFileSystemModel->setRootPath(mCurrPath);

getCurrentView()->setRootIndex(rootIndex);

}

// переключает текущий путь на указанный путь и обновляет представление файловой системы

bool MainWindow::jumpTo(const QString &path) {

if(path.compare(mFileSystemModel->myComputer().toString()) == 0) {

jumpToMyComputer();

return true;

}

QDir dir(path);

if(!dir.exists()) {

QMessageBox::critical(this, "File Explorer", ENTER\_DIR\_NOT\_EXIST\_MSG.arg(dir.absolutePath()), QMessageBox::Ok);

mEditTextView->setText(mCurrPath);

return false;

}

mCurrPath = dir.absolutePath();

mEditTextView->setText(mCurrPath);

QModelIndex index = mFileSystemModel->setRootPath(mCurrPath);

getCurrentView()->setRootIndex(index);

return true;

}

//метод возвращает текущий вид QAbstractItemView в зависимости от типа представления (Icon или Detail), который хранится в переменной mCurrType. Если тип представления Icon, возвращается указатель на mListView, если Detail - на TreeView

QAbstractItemView \*MainWindow::getCurrentView() {

switch (mCurrType) {

case Icon:

return mListView;

case Detail:

return mTreeView;

}

}

//Этот метод вызывается при нажатии Enter в поле ввода. Он переходит к пути, указанному в поле ввода, если это возможно, и добавляет новый путь в историю только в случае, если путь изменился

void MainWindow::onLineEditEnter() {

QString oldPath = mCurrPath;

if(jumpTo(mEditTextView->text()) && mCurrPath.compare(oldPath) != 0) {

mHistoryStack->push(mCurrPath);

}

}

//метод вызывается при нажатии кнопки "Вверх". Он переходит на уровень выше текущего каталога, если это возможно, и обновляет историю

void MainWindow::onUpButtonClick() {

if(isInMyComputer()) {

return;

}

QDir dir(mCurrPath);

if(dir.cdUp()) {

jumpTo(dir.absolutePath());

mHistoryStack->push(dir.absolutePath());

} else {

jumpToMyComputer();

mHistoryStack->push(mFileSystemModel->myComputer().toString());

}

}

//метод включает кнопку "назад"

void MainWindow::onBackButtonEnable(bool b) {

mBackButtonView->setEnabled(b);

}

//метод включает кнопку "вперед"

void MainWindow::onForwardButtonEnable(bool b) {

mForwardButtonView->setEnabled(b);

}

//вызывается при нажатии кнопки "назад"

void MainWindow::onBackButtonClick() {

QString backPath = mHistoryStack->backPath();

if(backPath.isEmpty()) {

return;

}

if(jumpTo(backPath)) {

mHistoryStack->back();

}

}

//вызывается при нажатии кнопки "вперед"

void MainWindow::onForwardButtonClick() {

QString forwardPath = mHistoryStack->forwardPath();

if(forwardPath.isEmpty()) {

return;

}

if(jumpTo(forwardPath)) {

mHistoryStack->forward();

}

}

//вызывается при двойном щелчке по элементу в представлении. Если элемент является директорией, то происходит переход в нее. Если элемент файл, то происходит попытка его открытия

void MainWindow::onItemDoubleClick(const QModelIndex& index)

{

if(!index.isValid()) {

return;

}

QFileInfo fileInfo = mFileSystemModel->fileInfo(index);

if(fileInfo.isDir()) {

jumpTo(fileInfo.absoluteFilePath());

mHistoryStack->push(fileInfo.absoluteFilePath());

} else {

QDesktopServices::openUrl(QUrl::fromLocalFile(fileInfo.absoluteFilePath()));

}

}

//вызывается при запросе контекстного меню (щелчок правой кнопкой мыши). Он определяет, где был произведен щелчок, и отображает соответствующее контекстное меню для файла или пустой области

void MainWindow::onCustomContextMenuRequested(const QPoint& point) {

if(isInMyComputer()) {

return;

}

if(getCurrentView()->indexAt(point).isValid()) {

QStringList list;

list.append(mFileSystemModel->filePath(getCurrentView()->indexAt(point)));

QModelIndexList selectList = getCurrentView()->selectionModel()->selectedIndexes();

for(auto index : selectList){

if(!list.contains(mFileSystemModel->filePath(index))) {

list.append(mFileSystemModel->filePath(index));

}

}

ITEMMENUINSTANCE->setItemList(list);

ITEMMENUINSTANCE->exec(QCursor::pos());

} else {

BLANKMENUINSTANCE->setPath(mCurrPath);

BLANKMENUINSTANCE->exec(QCursor::pos());

}

}

// вызывается при запросе обновления представления файловой системы. Он удаляет текущую модель файловой системы и создает новую, чтобы обновить данные. После этого обновляются представления

void MainWindow::onRefresh() {

delete mFileSystemModel;

mFileSystemModel = new FileSystemModel();

mListView->setModel(mFileSystemModel);

mTreeView->setModel(mFileSystemModel);

QModelIndex index = mFileSystemModel->setRootPath(mCurrPath);

mListView->setRootIndex(index);

mTreeView->setRootIndex(index);

}

//вызывается для удаления файла или директории по указанному пути. Если операция удаления не удалась, отображается сообщение об ошибке

void MainWindow::onRemove(const QString& path) {

if(!mFileSystemModel->remove(mFileSystemModel->index(path))) {

QMessageBox::critical(this, "Remove Error",

FILE\_REMOVE\_MSG.arg(path), QMessageBox::Ok);

}

if(isInNetwork()) {

onRefresh();

}

}

#include "RightClickMenu.h"

#include <QMimeData>

#include <QUrl>

#include <QApplication>

#include <QClipboard>

RightClickMenu::RightClickMenu(QWidget \*parent) : QMenu(parent) {};

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ItemMenu::ItemMenu(QWidget \*parent): RightClickMenu (parent) {

initWidget();

connect(mCutAction, &QAction::triggered,

this, &ItemMenu::onCutAction);

connect(mCopyAction, &QAction::triggered,

this, &ItemMenu::onCopyAction);

connect(mRemoveAction, &QAction::triggered,

this, &ItemMenu::onRemoveAction);

};

ItemMenu \*ItemMenu::instance() {

static ItemMenu\* ins = new ItemMenu();

return ins;

}

//Метод для установки списка выбранных элементов, которые будут обрабатываться контекстным меню

void ItemMenu::setItemList(const QStringList &list) {

mSelectedItemList = list;

}

//Обработчик события нажатия на пункт меню "Cut". Копирует выбранные элементы в буфер обмена с указанием типа операции "Cut"

void ItemMenu::onCutAction() {

QMimeData\* mimeData = new QMimeData;

QList<QUrl> urls;

for(QString item : mSelectedItemList) {

urls.append(QUrl::fromLocalFile(item));

}

mimeData->setUrls(urls);

int dropEffect = 2; // 2 for cut and 5 for copy

QByteArray data;

QDataStream stream(&data, QIODevice::WriteOnly);

stream.setByteOrder(QDataStream::LittleEndian);

stream << dropEffect;

mimeData->setData("Preferred DropEffect", data);

QApplication::clipboard()->setMimeData(mimeData);

}

//onCopyAct(): Обработчик события нажатия на пункт меню "Copy". Копирует выбранные элементы в буфер обмена с указанием типа операции "Copy"

void ItemMenu::onCopyAction() {

QMimeData\* mimeData = new QMimeData;

QList<QUrl> urls;

for(QString item : mSelectedItemList) {

urls.append(QUrl::fromLocalFile(item));

}

mimeData->setUrls(urls);

int dropEffect = 5; // 2 for cut and 5 for copy

QByteArray data;

QDataStream stream(&data, QIODevice::WriteOnly);

stream.setByteOrder(QDataStream::LittleEndian);

stream << dropEffect;

mimeData->setData("Preferred DropEffect", data);

QApplication::clipboard()->setMimeData(mimeData);

}

//Обработчик события нажатия на пункт меню "Delete". Генерирует сигнал sigRemove, чтобы удалить выбранные элементы

void ItemMenu::onRemoveAction() {

for(QString item : mSelectedItemList) {

emit sigRemove(item);

}

}

//Инициализирует виджет контекстного меню и его действия (пункты меню)

void ItemMenu::initWidget() {

mCutAction = new QAction(this);

mCopyAction = new QAction(this);

mRemoveAction = new QAction(this);

mCutAction->setText("Cut");

mCopyAction->setText("Copy");

mRemoveAction->setText("Delete");

addAction(mCutAction);

addAction(mCopyAction);

addAction(mRemoveAction);

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

BlankMenu::BlankMenu(QWidget \*parent) : RightClickMenu (parent) {

initWidget();

connect(mViewIconAction, &QAction::triggered,

this, &BlankMenu::onViewIconAction);

connect(mViewDetailAction, &QAction::triggered,

this, &BlankMenu::onViewDetailAction);

connect(mRefreshAction, &QAction::triggered,

this, &BlankMenu::sigRefresh);

connect(mPasteAction, &QAction::triggered,

this, &BlankMenu::onPasteAction);

}

BlankMenu\* BlankMenu::instance() {

static BlankMenu\* ins = new BlankMenu();

return ins;

}

//Устанавливает текущий путь в проводнике

void BlankMenu::setPath(const QString &info) {

mCurrPath = info;

}

//Обработчик события нажатия на пункт меню "Icon". Генерирует сигнал sigSwitchViewType, чтобы изменить тип отображения на иконки

void BlankMenu::onViewIconAction() {

emit sigSwitchViewType(MainWindow::Icon);

}

//Обработчик события нажатия на пункт меню "Detail". Генерирует сигнал sigSwitchViewType, чтобы изменить тип отображения на детали

void BlankMenu::onViewDetailAction() {

emit sigSwitchViewType(MainWindow::Detail);

}

//Обработчик события нажатия на пункт меню "Paste". Получает данные из буфера обмена и выполняет операцию вставки для выбранного пути

void BlankMenu::onPasteAction() {

const QMimeData\* mimeData = QApplication::clipboard()->mimeData();

if(!mimeData->hasUrls()) {

return;

}

if(!mimeData->hasFormat("Preferred DropEffect")) {

return;

}

int dropEffect = 0;

QByteArray data = mimeData->data("Preferred DropEffect");

QDataStream stream(&data, QIODevice::ReadOnly);

stream.setByteOrder(QDataStream::LittleEndian);

stream >> dropEffect;

QList<QUrl> urls = mimeData->urls();

if(dropEffect != 2 && dropEffect != 5) {

return;

}

QStringList srcs;

for(QUrl url : urls) {

if(!url.isLocalFile()) {

continue;

}

srcs << url.toLocalFile();

}

emit sigPaste(srcs, mCurrPath, dropEffect == 2);

if(dropEffect == 2) {

QApplication::clipboard()->clear();

}

}

//Инициализирует виджет контекстного меню и его действия (пункты меню), такие как изменение отображения и вставка

void BlankMenu::initWidget() {

mViewMenu = new QMenu(this);

mViewIconAction = new QAction(this);

mViewDetailAction = new QAction(this);

mPasteAction = new QAction(this);

mRefreshAction = new QAction(this);

mViewMenu->setTitle("View");

mViewIconAction->setText("Icon");

mViewDetailAction->setText("Detail");

mPasteAction->setText("Paste");

mRefreshAction->setText("Refresh");

mViewMenu->addAction(mViewIconAction);

mViewMenu->addAction(mViewDetailAction);

addMenu(mViewMenu);

addAction(mRefreshAction);

addAction(mPasteAction);

}

ПРИЛОЖЕНИЕ Д

(обязательное)

**Ведомость документов**