Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут деревообробних та комп'ютерних технологій і дизайну
(повне найменування інституту, назва факультету (відділення))

Кафедра інформаційних технологій
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

другий (магістерський)
(рівень вищої освіти)

на тему: «Розроблення кросплатформного менеджера файлової системи для роботи з хмарним середовищем».

Виконав: студент 6 курсу групи КН(М)-61
спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Волянський В. С.
(прізвище та ініціали)

Керівник Борецька І. Б.
(прізвище та ініціали)

Рецензент Лесьо А. П.
(прізвище та ініціали)

Львів – 2022

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ деревообробних та комп'ютерних технологій і дизайну
Кафедра інформаційних технологій
Рівень вищої освіти другий (магістерський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Крошний І. М.
"____" _____ 20___ року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Волянському Віталію Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи: «Розроблення кросплатформного менеджера файлової системи для роботи з хмарним середовищем»

керівник роботи Борецька Ірина Богданівна, кандидат технічних наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "13" грудня 2021 року № С-617

2. Термін подання студентом роботи 02.12.2022 р.

3. Вихідні дані до роботи Офіційна документація бібліотеки Qt та інша література, наведена у списку використаних джерел

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) аналіз ринку файлових менеджерів, розробка власного рішення у цій сфері, оцінка додатку за загальноприйнятим параметрам.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
MVC архітектура, UML діаграма класів, користувацький потік, прототип додатку

6. Дата видачі завдання 20.12.2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Наказ на затвердження тем - № С-617	13.12.2021	
2	Видача завдання	20.12.2021	
3	Системний аналіз стану проблемної області. Огляд літературних джерел згідно досліджуваної теми.	21.12.2021	
4	Визначити архітектуру для розроблюваного програмного забезпечення, та засоби для його розробки	01.02.2022	
5	Створити інтерактивний прототип зі всіма вікнами та стилями	11.04.2022	
6	Розробити працюючу програму	13.06.2022	
7	Провести необхідні виміри по різних критеріях	10.10.22	
8	Термін здачі пояснювальної записки на кафедру	12.12.2022	
9	Захисти дипломних робіт	27.12.2022	

Студент _____ Волянський В.С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Борецька І.Б.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 60 сторінок пояснювальної записки, 30 рисунків, 11 таблиць, 1 додаток, 13 джерел.

В даній роботі розроблений кросплатформовий двохпанельний файловий менеджер. Даний додаток дозволяє виконувати всі базові операції з файлами, підтримує багато інших функцій наявних в сучасних аналогах, а також працює на кількох операційних системах.

Об'єкт дослідження: файлова система та програмне забезпечення для управління нею.

Мета роботи: розробка власного рішення у сфері файлових менеджерів.

Значущість роботи: зручна та швидка робота з файловою системою може спростити життя багатьом технічним спеціалістам.

Ключові слова: Qt, GUI, Файлові системи, багатоплатформність, операційні системи, файловий менеджер.

ABSTRACT

The thesis contains 60 pages of explanatory note, 30 figures, 11 tables, 1 appendix, 13 used literary sources.

In this work, a cross-platform two-panel file manager was developed. This application allows you to perform all basic operations with files, supports many other features available in modern analogues, and also works on several operating systems.

The object of research: the file system and the software for managing it.

The goal of the work: development of own solution in the field of file managers

Significance of work: convenient and fast work with the file system can simplify the life of many technical specialists.

Keywords: Qt, GUI, File systems, cross-platform, operating systems, file manager.

ТЕХНІЧНЕ ЗАВДАННЯ

Необхідно розробити програмне забезпечення, що буде виконувати роль файлового менеджера, а саме:

- Проаналізувати ринок на предмет існуючих файлових менеджерів і визначити необхідний функціонал;
- Визначити архітектуру для розроблюваного програмного забезпечення, та засоби для його розробки;
- Створити інтерактивний прототип зі всіма вікнами та стилями;
- Розробити працюючу програму;
- Провести необхідні виміри по різних критеріях (швидкодія, розмір на диску, споживані системні ресурси).

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

OS – Operating System
FTP – File Transfer Protocol
CD – Compact Disc
DVD – Digital Video Disc
CPU – Central Processing Unit
RAM – Random Access Memory
SSD – Solid-State Drive
NTFS – New Technology File System
TLC – Triple-Level Cell
DDR – Double Data Rate
SPD – Serial Presence Detect
CLI – Command Line Interface
API – Application Programming Interface
MVC – Model View Controller

ВСТУП

Зберігання інформації – одна з найважливіших функцій будь-якої ОС. Тому існують такі поняття як файл і файлова система. Коротко кажучи, файли – це інформаційні об'єкти, які містять дані або програми, а файлова система, відповідно, є способом організації цих об'єктів.

Однак, оскільки користувачами операційної системи часто є люди, недостатньо зберігати інформацію, необхідно ще й забезпечити зручний спосіб роботи з нею. Ось тут і вступають у гру файлові менеджери. Ми використовуємо їх майже кожного разу, коли користуємося комп’ютером, і тому вони надзвичайно потрібні бути швидкими та зручними.

Менеджери файлів також використовуються в різноманітних технічних напрямах, таких як програмне забезпечення та веб-розробка, графічний дизайн, анімація, редагування відео, розробка мереж, аналіз даних тощо. Менеджери файлів є критично важливими для таких завдань, оскільки вони забезпечують швидкий доступ до необхідних цифрових активів і пропонують додаткові організаційні інструменти, такі як створення папок і вкладених папок для кращої організації.

Крім цього, файлові менеджери пропонують такі заходи безпеки, як налаштування захисту паролем, шифрування, права доступу, процеси резервного копіювання та відновлення.

Це і є головною темою даної роботи.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

Файловий менеджер — це комп’ютерна програма, що забезпечує інтерфейс користувача для роботи з файловою системою та файлами. Файловий менеджер дозволяє виконувати найпоширеніші операції з файлами — створення, відкриття, редагування, переміщення, перейменування, копіювання, видалення, зміна атрибутів і властивостей, пошук файлів і призначення прав. На додаток до основних функцій багато файлових менеджерів включають низку додаткових функцій, таких як мережеві функції, резервне копіювання, керування принтером тощо.

Однак для людей, які мають досвід роботи в цій області, очевидно, що сучасні файлові менеджери далекі від ідеалу. Деякі не мають достатнього функціоналу, для фахівців окремих сфер. Одразу спадають на думку стандартні рішення, надані розробниками операційних систем, наприклад Windows Explorer або Nautilus для Ubuntu. Є й інша крайність — перенасичення функціоналом, що в свою чергу впливає на інші характеристики, наприклад на зручність використання. Прикладом може служити улюблений багатьма менеджер Total Commander. Однак, коли користувач встановлює його, він відразу губиться в багатьох кнопках і функціях, якими він не планує користуватися. Критичним недоліком деяких менеджерів є проблеми з дизайном UI/UX. Розробники намагаються вкласти в свій проект все те, що є у конкурентів, але не помічають, що їхнім продуктом практично неможливо користуватися.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Файлова система

Файлова система — порядок, що визначає спосіб організації, зберігання та іменування даних на носіях в комп'ютерах, а також в іншому електронному обладнанні. Файлова система визначає формат вмісту та фізичного зберігання інформації, яка групується у файли. Файлова система з’єднує носій даних з одного боку та API доступу до файлів з іншого. Коли програма звертається до файлу, вона не має уявлення про те, як у конкретному файлі розташована інформація, а також на якому фізичному носії (CD, жорсткий диск, магнітна стрічка, блок флеш-пам’яті) вона записана. Програма знає тільки назву файлу, його розмір і атрибути. Вона отримує ці дані від драйвера файлової системи. Саме файлова система визначає, де і як файл буде записаний на фізичний носій (наприклад, жорсткий диск).

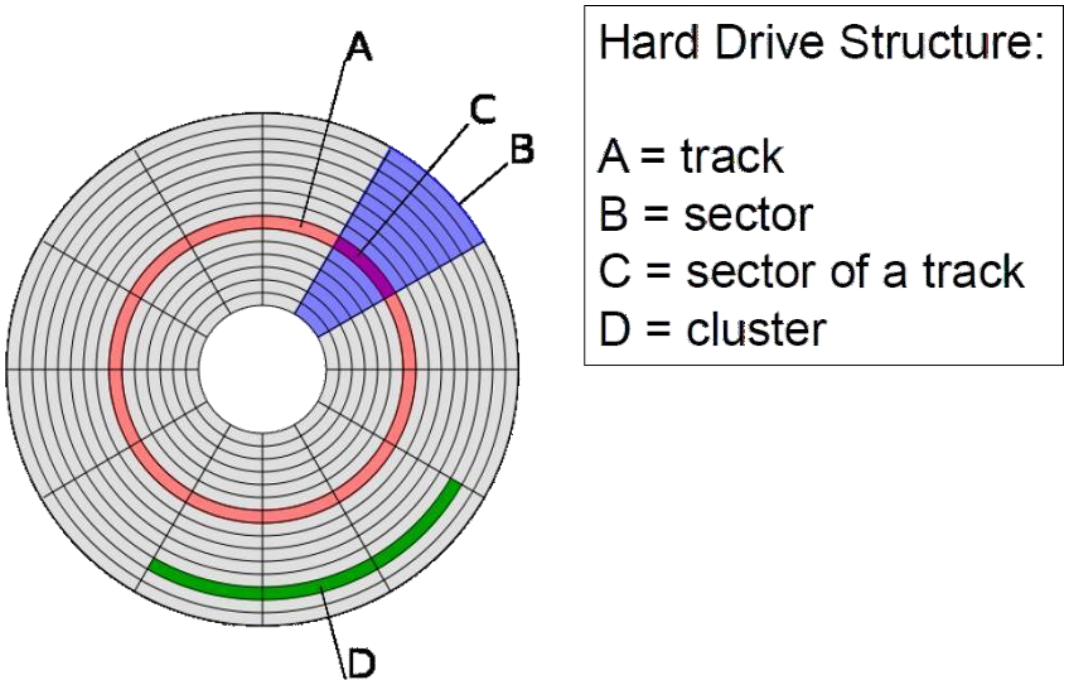

Рисунок 2.1. Структура жорсткого диску.

З точки зору операційної системи, весь диск є набором кластерів (зазвичай 512 байт або більше). Драйвери файлової системи організовують кластери у файли та

директорії (які насправді є файлами, що містять список файлів у цій директорії). Ті самі драйвери відстежують, які з кластерів зараз використовуються, які вільні, а які позначені як несправні. Окрім файлів користувача, файлова система також містить власні метадані (такі як розмір блоку), дескриптори файлів (розмір файлу, розташування, фрагменти тощо), імена файлів та ієрархію каталогів. Вона також може зберігати інформацію про безпеку, розширені атрибути та інші налаштування. В основному, це залежить від типу файлової системи, яких дуже багато навіть в умовах однієї операційної системи.

Рисунок 2.2. Порівняння файлових систем.

2.2. Головні архітектурні принципи

Розроблена програма побудована за шаблоном MVC. Вона розділена на три окремі частини. Model відповідає за зберігання даних і їх структуру. View відповідає за представлення цих даних користувачеві, тобто програмний інтерфейс. Controller керує компонентами, отримує сигнали у відповідь на дії користувача (зміна

положення курсору миші, натискання кнопки, введення даних у текстове поле) і передає дані моделі.

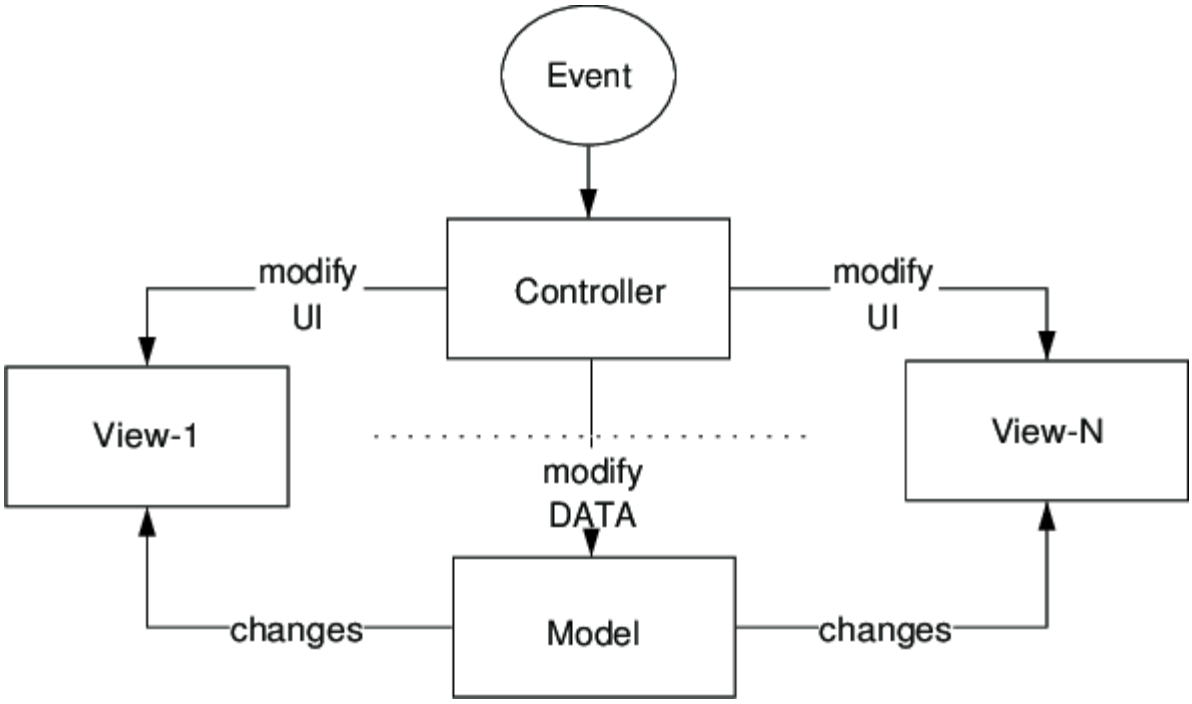

Рисунок 2.3. MVC архітектура.

Роль моделі відіграє різна бізнес-логіка та представлення файлової системи. Для отримання програмного доступу до файлової системи використовувалася бібліотека Qt, а саме клас QFileSystemModel. Цей клас забезпечує доступ до локальної файлової системи та різні способи взаємодії з нею. Він досить добре оптимізований з точки зору продуктивності. Він використовує QFileSystemWatcher, щоб автоматично підтримувати всю інформацію в актуальному стані. View представлена структурою віджетів Qt, заздалегідь описаною у спеціальних UI формах. Усі ці віджети успадковують клас QWidget і можуть використовуватися для відображення різних даних, отримання введених даних користувачами, а також служать контейнером для інших віджетів. Як згадувалося вище, view підключається до моделі за допомогою контролера. В даному випадку це сигнали. Вони запускаються, коли користувач виконує певну дію. Кожен сигнал має певний слот

(звичайна функція C++), підключений до нього. При спрацьовуванні сигналу виконується відповідний слот.

2.3. Використані засоби

Таблиця 2.1

	Назва	Версія	Застосування
Мова програмування	C++	11	Служить для надання базового синтаксису та визначення поведінки програми
Система автоматизації збірки проекту	CMake	3.19.2	Використовується для автоматизації створення програмного забезпечення з вихідного коду. Перевіряє наявність необхідних бібліотек і підключає їх, збирає проект під різні компілятори та операційні системи.
Компілятор	MinGW g++	8.1.0	Перетворює вихідний код, написаний на C++, у семантично еквівалентний машинний код, необхідний для запуску програми комп’ютером.
Фреймворк для розробки програмного забезпечення	Qt	5.15.2	Один із найзручніших фреймворків для розробки програм із графічним інтерфейсом користувача. Qt надає не лише відповідний набір бібліотек класів, а й конкретну модель розробки додатків. Важливою перевагою Qt є продуманий і логічний набір класів, що забезпечує високий рівень абстракції. Крім того, Qt є кросплатформенним, і щоб запустити програму на іншій ОС, вам просто потрібно перекомпілювати вихідний код.

Бібліотека для побудови діаграм	Qt Charts	5.15.2	Надає набір простих у використанні компонентів діаграми. Вона використовує Qt Graphics View Framework, тому діаграми можна легко інтегрувати в інтерфейс користувача.
IDE для написання коду	Qt Creator	4.14.0	Найкраще середовище розробки для Qt. Має власну підтримку форм інтерфейсу користувача, а також вбудований інструмент створення інтерфейсу користувача.
IDE для створення інтерфейсу	Qt Designer	4.14.0	Використовується для створення та налаштування форм конструктора Qt (*.ui). Ці форми представляють дерево віджетів у форматі XML (QML) і будуть перетворені на код C ++, який можна скомпілювати. Таким чином ви можете набагато гнучкіше налаштовувати зовнішній вигляд віджета, якщо він має складну структуру і немає сенсу вбудовувати його безпосередньо в код.

2.4. Архітектура програми

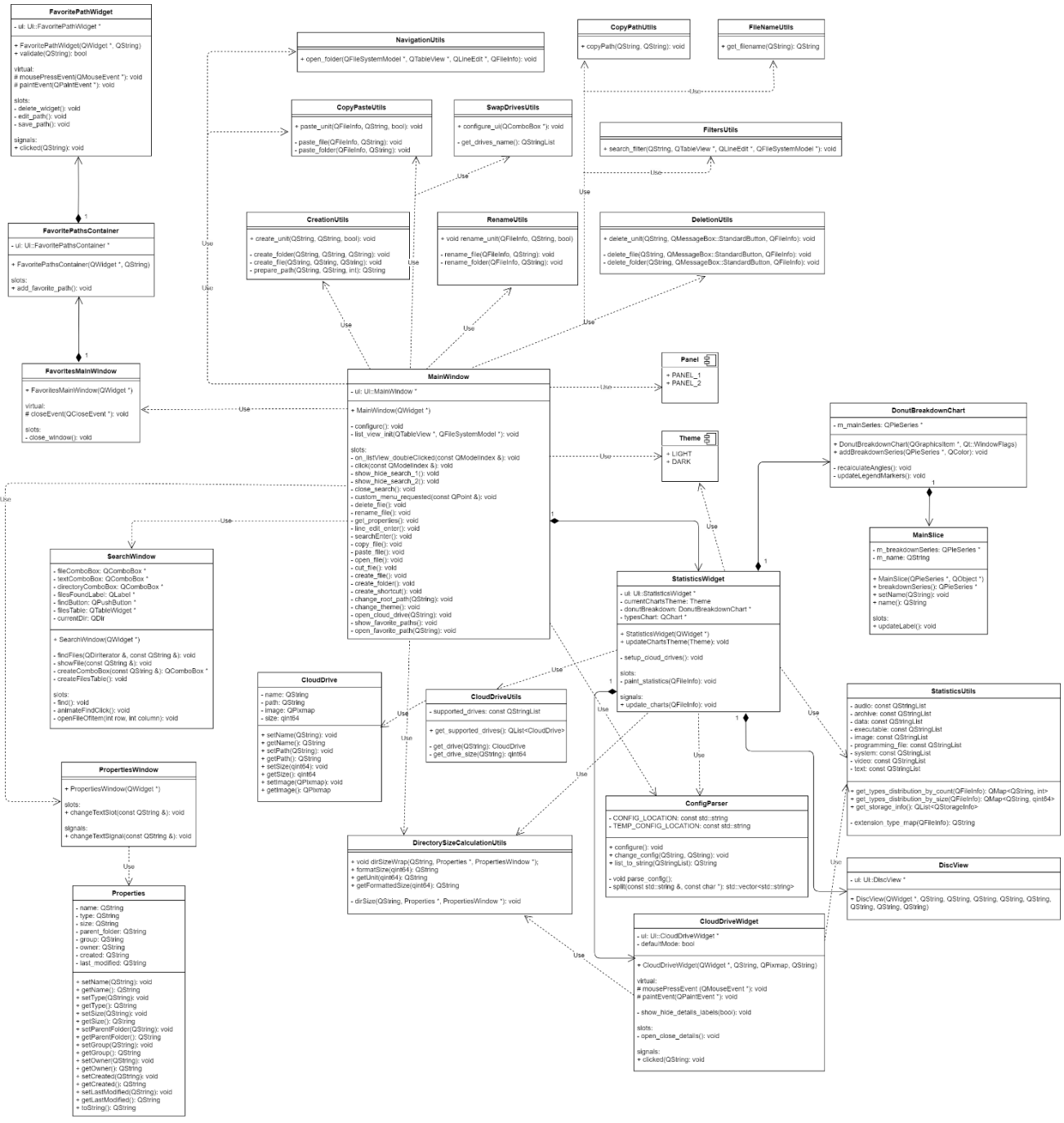

Рисунок 2.4. UML діаграма класів.

РОЗДІЛ 3. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Головний функціонал

3.1.1. Операції з файлами

1. Відкриття.

Опис. Можливість відкривати різні елементи у файловій системі. Щоб відкрити файл, потрібно двічі клацнути по ньому або вибрати відповідний пункт у контекстному меню. Після цього файл буде відкрито за допомогою програми за замовчуванням.

Застосування. Читання та запис файлів, а також навігація деревом файлової системи є однією з найважливіших функцій будь-якого файлового менеджера.

Реалізація. Відкриття файлів базується на функції `bool QDesktopServices::openUrl(const Qurl &url)` Ми отримуємо абсолютний шлях до файлу за допомогою `Qstring QfileInfo::absoluteFilePath()` і вставляємо його як аргумент. Ця дія підключається до двійного кліку мишкою та відповідного пункту контекстного меню. Відкриття папки реалізовано за замовчуванням під час підключення `QtableView` та `QfileSystemModel`.

2. Створення.

Опис. Для цього в контекстному меню потрібно вибрати відповідний пункт, а потім ввести назву. Елемент з’явиться в поточному каталозі. Завдяки мультивибору стало можливим створити папку з попередньо визначеним вмістом. Для цього виберіть кілька пунктів, викличте контекстне меню і створіть папку.

Застосування. Створення нових вузлів і елементів є однією з важливих функцій файлової системи, якою також необхідно керувати.

Реалізація. Створення файлів реалізовано за допомогою

```
bool Qfile::open(int fd, QIODevice::OpenMode mode, QfileDevice::FileHandleFlags handleFlags = DontCloseHandle)
```

Після виклику цієї функції Qt створить файл, якого перед виконанням дії не існувало, і після операції він буде автоматично закритий. У випадку, якщо файл з даним іменем існував, до нього додається лічильник. Наприклад file(1).txt.

3. Копіювання.

Опис. Позначає елемент для копіювання. Це потрібно для того, щоб точно знати, що потрібно переміщати/вирізати в майбутньому.

Застосування. Часто користувачеві може знадобитися дублікат папки або файлу. Оригінальна версія при цьому буде збережена.

Реалізація. Оскільки це лише маркування, тут немає складної логіки. Усі виділені елементи поміщаються в глобальний буфер, звідки вони будуть взяті у разі вставки.

4. Вирізання.

Опис. Вирізання — це те ж саме, що і копіювання, за винятком того, що оригінал не зберігається. Після вирізання матимемо лише одну копію елемента, але в цільовому каталозі замість початкового.

Застосування. Вирізання – фактично зміна каталогу. Таким чином, користувач може переміщувати елемент у дереві файлової системи.

Реалізація. Відбувається те ж саме, що і при копіюванні з однією відмінністю. Наприкінці цього процесу глобальній змінній `bool toCut` присвоюється значення `true.`

5. Вставка.

Опис. Вставка безпосередньо пов’язана з копіюванням і вирізанням. Якщо на попередньому кроці ми позначили файл/папку для копіювання/вирізання, то тепер ми виконуємо цю операцію, вказуючи цільовий каталог. Після цього там з’являться елементи.

Застосування. Вибрати цільовий каталог і підтвердити виконання операції вирізання/копіювання.

Реалізація. У випадку файлів використовується наступна функція `bool Qfile::copy(const Qstring &fileName, const Qstring &newName)`. Вона копіює файл із вихідного розташування (1-й аргумент) у місце призначення (2-й аргумент). З папками все трохи складніше. Необхідно двічі рекурсивно перебирати початковий каталог. Під час першого разу ми створюємо ту саму структуру каталогу в цільовому місці, що й у вихідному. Щоб отримати докладнішу інформацію про створення, див 3.1.2. Під час другого разу – всі файли по черзі копіюються у відповідні місця, створені під час попередньої ітерації. Після цього перевіряється глобальна змінна `bool toCut`. У випадку `true` файл/папка видаляється з місця призначення. Щоб дізнатися більше про видалення, див 3.1.6.

6. Видалення.

Опис. Видаляє файли або цілі папки з файлової системи. Після цього пам'ять, яку вони займали на жорсткому диску, буде звільнена.

Застосування. Очистити пам'ять, зайняту непотрібними файлами.

Реалізація. Перед початком видалення виконуються різноманітні перевірки доступу та прав. Якщо ці перевірки не проходять успішно, буде показано відповідне попередження. У разі успіху файл/папка буде видалена. Функції видалення надаються Qt за замовчуванням. Для файлів це `bool Qfile::remove()`, для папок — `bool Qdir::removeRecursively()`.

7. Перейменування.

Опис. Ім'я файлу – це символьний рядок, який унікально ідентифікує файл/папку в каталозі. Імена файлів будуються за правилами, прийнятими в операційній системі. Імена файлів також використовуються для побудови абсолютного/відносного шляху, для отримання доступу до них будь-де. Ця функція дозволяє змінити такий ідентифікатор.

Застосування. Перейменування файлів важливо для кращого розуміння того, що являє собою кожен файл. Таким чином можна приблизно оцінити його вміст, навіть не відкриваючи його.

Реалізація. Перед перейменуванням файлу перевіряється, чи може користувач редагувати його: `info.permission(Qfile::WriteUser)`

Після проходження цієї перевірки файл буде перейменовано за допомогою функцій:

```
bool Qfile::rename(const Qstring &newName);
bool Qdir::rename(const Qstring &oldName, const Qstring &newName);
```

Випадок, коли такий файл/папка вже існує, опрацьовується так само, як і при створенні, див. 3.1.2.

8. Ярлики

Опис. З точки зору файлової системи, ярлик — це дескриптор, який дозволяє користувачеві знаходити файл або ресурс, розташовані в іншому каталозі чи папці. Функція дозволяє створювати ярлики для файлів і папок. Коли ви відкриваєте ярлик файлу, відкриється відповідний файл, коли ви відкриваєте ярлик папки, поточний каталог на активній панелі зміниться.

Застосування. Іноді файл може бути сильно прив’язаний до свого каталогу, і його розташування неможливо змінити. Однак користувач часто взаємодіє з цим файлом і хоче робити це зі зручного місця, не змінюючи розташування файлу. Для забезпечення цієї можливості потрібні ярлики. Вони надають швидкий доступ до будь-якого елемента файлової системи.

Реалізація. Створення ярликів базується на функції `bool Qfile::link(const Qstring &linkName)`. Ця функція викликається, коли користувач хоче створити ярлик для певного файлу. Аргументи — це абсолютний шлях до файлу та той самий шлях, але з розширенням .lnk, яке насправді є розширенням ярликів. Для папок процес ідентичний.

9. Властивості.

Опис. Вікно з детальною інформацією про обраний елемент. Ця інформація містить такі дані: ім’я файлу, тип (файл, ярлик, каталог тощо), розмір, батьківська папка, група (системи Unix), власник (системи Unix), дата створення цього елемента, дата останньої зміни.

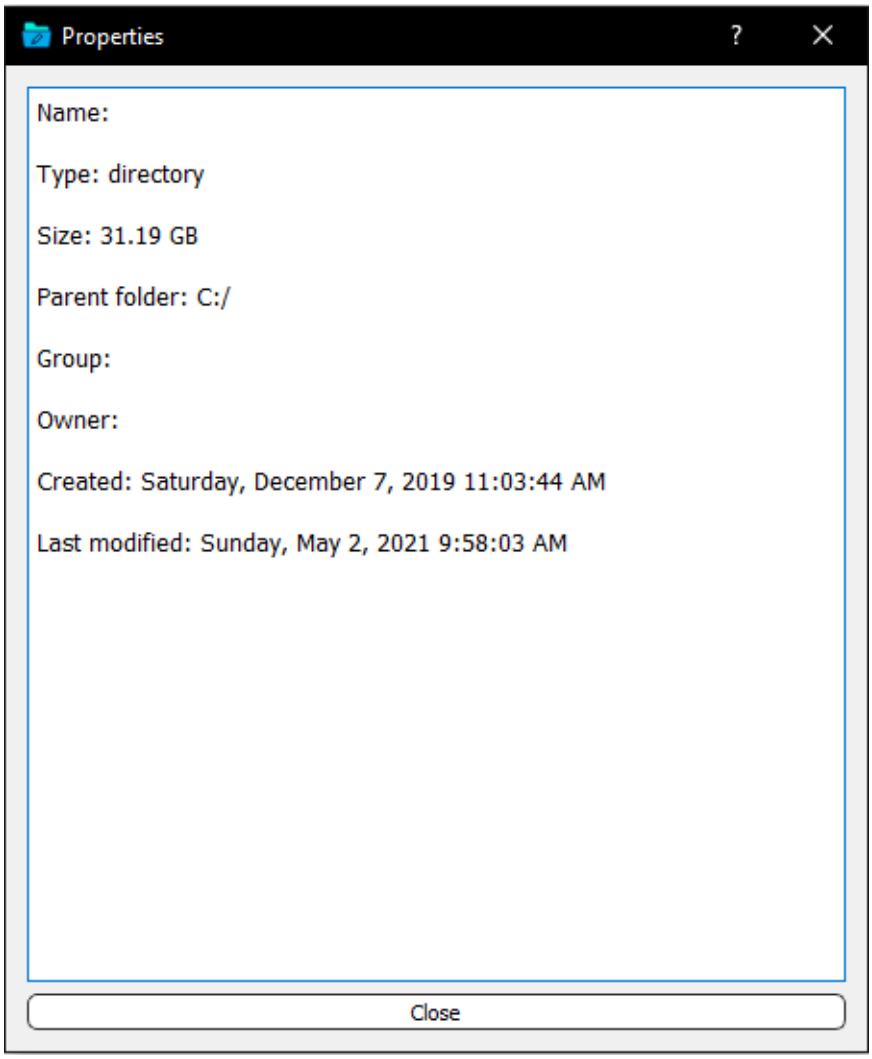

Рисунок 3.1. Вікно властивостей.

Застосування. Користувачеві може знадобитися додаткова інформація про файл. Такі речі, як розмір, абсолютний шлях і тип, можуть бути надзвичайно корисними.

Реалізація. Звичайне спливаюче вікно, заповнене вмістом класу `QfileInfo`.

3.1.2. Фільтри

Опис. Фільтрація в додатку реалізована за допомогою спеціального поля. У цьому полі можна ввести як повну назву файлу, так і шаблон, за яким його потрібно шукати. Підтримуються регулярні вирази.

Застосування. Швидка та проста заміна глобального пошуку, але з деякими обмеженнями. Економить час, коли вам потрібно знайти файл за його назвою в одному каталозі.

Реалізація. Коли користувач підтверджує свій ввід, запускається спеціальна функція. На основі цих вхідних даних формується запит, після чого до моделі файлової системи застосовуються відповідні фільтри.

```
  QstringList filters;
1
  Qstring request = «*»;
2
  request.append(item_name);
3
  request.append(«*»);
4
  filters << request;
5
  filters << «.»;
6
  filters << «..»;
7
  model->setNameFilters(filters);
8
```

3.1.3. Сортування

Опис. Сортування реалізовано за допомогою шапки, розташованої у верхній частині кожної панелі. Ви можете натиснути категорію, яка вас цікавить, і відсортувати всі записи в порядку зростання/спадання.

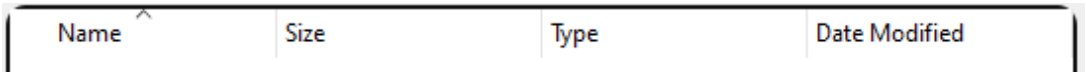

Рисунок 3.2. Секція сортування.

Застосування. Сортування може бути корисним у багатьох ситуаціях. Наприклад, знайти у папці найбільший/останній змінений/найстаріший файл тощо.

Реалізація. Для представлення файлової системи використовується `QtableView`. За замовчуванням він має заголовок, який насправді є класом `QheaderView`. Щоб мати можливість використовувати його для сортування, потрібна лише базова конфігурація.

```
tableView->horizontalHeader()->setStretchLastSection(true);
tableView->horizontalHeader()->setSortIndicator(0, Qt::AscendingOrder);
tableView->horizontalHeader()->setSectionResizeMode(0, QheaderView::Stretch);
```

3.1.4. Рядок навігації

Опис. Рядок навігації служить для відображення та контролю розташування користувача в дереві файлової системи. Він містить абсолютний шлях поточного каталогу. Кожного разу, коли він змінюється, текст рядка також оновлюється. Крім того, ви можете ввести свій шлях і якщо він дійсний, то, натиснувши відповідну кнопку, ви перейдете в потрібний каталог. Кожна панель має свій рядок навігації.

Застосування. Використовується для відображення абсолютного шляху поточного каталогу, а також для швидкого переходу до заданого користувачем шляху.

Реалізація. Реалізовано за допомогою віджета `QlineEdit`. Коли користувач натискає Enter на клавіатурі, введенні дані проходять валідацію і відбувається перехід до вказаного шляху. Крім того, щоразу, коли каталог змінюється, новий абсолютний шлях записується у відповідний рядок навігації.

3.1.5. Глобальний пошук

Опис. Дозволяє шукати файл у файловій системі за такими критеріями: назва файлу, текстовий вміст, батьківський каталог. Коли пошукова система знайде файли, вони з'являться в списку, де їх можна буде відкрити. Не підтримує бінарні файли та архіви.

Застосування. Потужний механізм, який дозволяє легко знайти будь-що у файловій системі.

Реалізація. Після підтвердження користувачем свого введення дані, отримані з трьох полів, перевіряються та обробляються. Якщо поле шляху порожнє, використовується кореневий каталог. Якщо ім'я файлу не вказано — пошукова система його не враховує і файл з будь-яким ім'ям є метою пошуку. Те ж саме відбувається, коли поле тексту не заповнене. Щоб перебирати каталоги з метою пошуку цільових файлів, використовується `QdirIterator`:

```
QdirIterator it(path, filter, Qdir::AllEntries | Qdir::NoSymLinks |
Qdir::NoDotAndDotDot, QdirIterator::Subdirectories);
```

Після цього в новому потоці запускається функція пошуку за допомогою `std::thread` з паралельної бібліотеки C++. Мета полягає в тому, щоб не зупиняти основний потік додатків таким довготривалим процесом. Файли з'являтимуться у вікні, коли вони будуть знайдені, і користувач зможе взаємодіяти з ними, навіть якщо процес пошуку ще не завершено. Як згадувалося вище, виконується ітерація по каталогах. У випадку, якщо користувач хоче шукати файл за текстовим вмістом, потрібні додаткові операції. Якщо файл відповідає всім попереднім параметрам, механізм повинен перевірити його на наявність вказаного тексту. Для цього файл потрібно відкрити та просканувати за допомогою `QtextStream`:

```
 1  Qfile file(fileName);
 2  if (file.open(QIODevice::ReadOnly)) {
 3      Qstring line;
 4      QtextStream in(&file);
 5      while (!in.atEnd()) {
 6          line = in.readLine();
 7          if (line.contains(text, Qt::CaseInsensitive)) {
 8              foundFiles << files[i];
 9              break;
10          }
11      }
12  }
```

Коли всі файли знайдено або вікно закрито, процес пошуку завершується.

3.1.6. Прикріплені папки

Опис. Функція збереження та переходу до часто використовуваних каталогів. При бажанні користувач може зберігати часто відвідувані каталоги, редагувати їх і видаляти, якщо вони більше не потрібні.

Застосування. Часто важливі папки можуть бути розташовані дуже глибоко і тому незручні для доступу. Щоб спростити життя користувача та заощадити його час, реалізовано цю функцію. Основна перевага перед ярликами полягає в тому, що список улюблених шляхів доступний будь-де, оскільки він не прив'язаний до файлової системи. Таким чином, користувачеві не потрібно переходити в папку з ярликами, а просто натиснути одну кнопку.

Реалізація. Головну роль у реалізації цієї функції відіграють два базові віджети: `FavoritePathWidget` і `FavoritePathWidgetContainer`. Як видно з їх назв, перший відповідає обраному шляху і може бути доданий користувачем, інший служить контейнером для шляхів і може бути викликаний відповідною кнопкою.

Коли користувач намагається додати новий шлях, спочатку він проходить просту перевірку.

```
1  if (Qdir(path).exists() && !path.isEmpty()) {
2      return true;
3  }
```

Після успішної перевірки віджет створюється та додається до контейнера. До цього віджету підключений слот з відкриттям відповідної папки. Щоб зберегти шлях навіть після закриття програми, її потрібно додати до зовнішньої пам’яті. У нашому випадку це просто конфігураційний файл, в якому записане рядкове представлення шляху. Крім того, є локальний кеш, який завантажує всі записи, щоб не зчитувати їх кожного разу з файлу. Таким чином, усі взаємодії проходять через цей кеш і лише після закриття віджета зміни застосовуються до основного сховища.

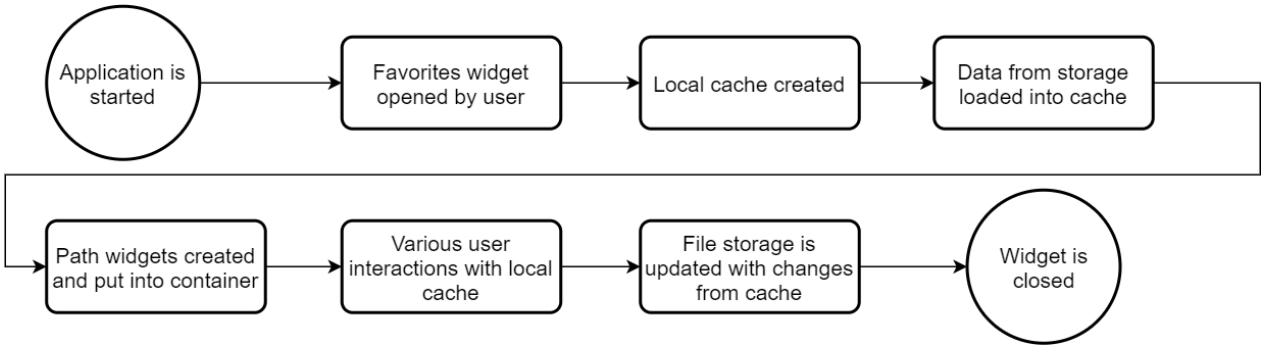

Рисунок 3.3. Життєвий цикл віджета прикріплених папок.

З точки зору інтерфейсу користувача немає обмежень на кількість записів, оскільки вони розташовані в `QscrollArea`, і коли їх більше не можна буде відображати в контейнері, автоматично буде додано повзунок.

3.1.7. Підтримка хмарних сховищ

Опис. Віджет, який дозволяє швидко відкрити ваше хмарне сховище, а також надає додаткову інформацію про нього: ім’я користувача, використану пам’ять,

розподіл типів. Підтримує не лише OneDrive, але й будь-яке інше сховище, якщо воно має локальне представлення на пристрої.

Застосування. Хмарне сховище – це модель зберігання даних у комп'ютері, за якої цифрові дані зберігаються в логічних пулах, а фізичне сховище охоплює кілька серверів (зазвичай у кількох місцях). Хмарне сховище надає користувачам миттєвий доступ до широкого спектру ресурсів і програм, розміщених в інфраструктурі іншої організації, через інтерфейс веб-сервісу. Основна перевага полягає в тому, що воно не прив'язане до чогось локального, ви можете використовувати його з будь-якого пристрою в будь-якому місці. Останнім часом ринок хмарних сховищ демонструє величезне зростання.

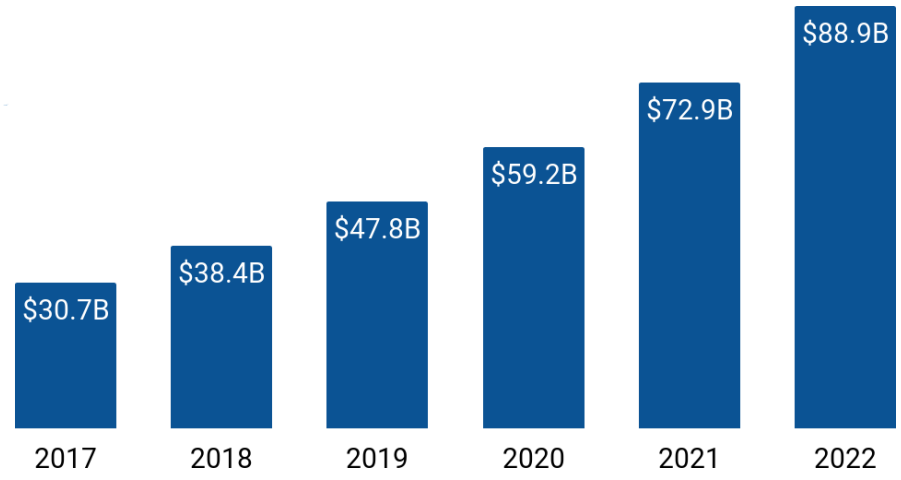

Рисунок 3.4. Зростання ринку хмарних сховищ.

Реалізація. Як згадувалося вище, кожне хмарне сховище має власне локальне представлення на пристрої користувача. Зазвичай вони знаходяться в домашній директорії. Щоб отримати міжплатформний доступ до цього розташування: `Qstring Qdir::homePath()`. Після цього починається налаштування віджета. Ми створюємо сигнал клацання та надсилаємо його у випадку

```
void CloudDriveWidget::mousePressEvent(QmouseEvent *event).
```

Відкриття папки на локальному диску підключається до цього сигналу. Крім того, додаткова інформація витягується за допомогою класу `QfileInfo`, а також

механізму розподілу типів. Маючи цю інформацію, ми можемо заповнити нею віджет, після чого він буде готовий до використання.

3.1.8. Інформаційна панель

Інформаційна панель, розташована в правій частині інтерфейсу, відображає різну інформацію про файлову систему, каталог, де знаходиться користувач, наявні диски тощо. Вона містить три віджети:

а. Загальна діаграма розподілу пам’яті

Опис. Кругова діаграма поділена на дві частини: зовнішню і внутрішню. Внутрішня показує, як пам’ять розподіляється між віртуальними дисками. Зовнішня показує розподіл між зайнятою і вільною пам’яттю для кожного з дисків.

Застосування. Користувач повинен завжди бути в курсі стану своєї пам’яті. Таким чином можна визначити, який диск має найвищий пріоритет для використання, наскільки великі файли він може зберігати та коли очищати пам’ять для майбутнього використання.

Реалізація. В якості основи для побудови діаграм була використана бібліотека Qt Charts. Інформацію про стан пам’яті взято з класу `QstorageInfo`:

```
qint64 QstorageInfo::bytesTotal();
qint64 QstorageInfo::bytesFree();
```

Маючи цю інформацію, можна легко побудувати загальну діаграму розподілу пам’яті:

```
1 DonutBreakdownChart chart = new DonutBreakdownChart();
2 for (QstorageInfo info : infoList) {
3     QpieSeries *series = new QpieSeries();
4     series->setName(info.rootPath());
5     series->append(«Free», info.bytesAvailable());
6     series->append(«Used», info.bytesTotal() – info.bytesAvailable());
7     donutBreakdown->addBreakdownSeries(series);
8 }
```

b. Віджет з інформацією про диски

Опис. Віджет показує детальнішу інформацію про кожен із дисків, присутніх в операційній системі, включаючи ім’я, ємність, тип файлової системи, чи підключений диск і чи є він лише для читання.

Застосування. Є багато випадків використання, де це може бути корисним. Розмонтовані диски не можна використовувати, доки вони не будуть повторно змонтовані вручну. Диски лише для читання не можуть приймати жодних змін. Багато факторів також залежать від типу файлової системи.

Реалізація. Віджет інформації про диски представлено `QtabWidget` із n вкладками, де n дорівнює кількості дисків. Щоб отримати інформацію про всі диски: `Qlist<QstorageInfo> QstorageInfo::mountedVolumes()`. Після цього ми переглядаємо цей список, витягуємо необхідну інформацію з кожного запису (диска) і заповнюємо нею віджет.

c. Діаграма розподілу типів

Опис. Секторна діаграма має динамічну кількість фрагментів. Кожен із цих фрагментів представляє певний тип файлу.

Таблиця 3.1

Тип	**Розширення**
Аудіо	aif, cda, mid, midi, mp3, mpa, ogg, wav, wma, wpl
Архів	7z, arj, deb, pkg, rar, rpm, gz, z, zip
Дані	csv, dat, db, log, mdb, sav, sql, xml
Виконуваний файл	apk, bat, bin, com, exe, gadget, jar, msi, wsf
Зображення	ai, bmp, gif, ico, jpeg, jpg, png, ps, psd, svg, tif, tiff
Файл мови програмування	c, cgi, pl, class, cpp, cs, h, java, php, py, sh, swift, vb

Системний файл	bak, cab, cfg, cpl, cur, dll, dmp, drv, icns, ico, ini, lnk, msi, sys, tmp
Відео	3g2, 3gp, avi, flv, h264, m4v, mkv, mov, mp4, mpg, mpeg, rm, swf, vob, wmv
Текст	doc, docx, odt, pdf, rtf, tex, txt, wpd

Таким чином можна оцінити розподіл типів у поточному каталозі. Коли каталог змінюється, діаграма оновлюється.

Застосування. Може бути корисним, щоб охарактеризувати каталог з точки зору існуючих типів файлів, перевірити наявність додаткових типів і швидко зрозуміти, для чого він використовується.

Реалізація. В якості основи для побудови діаграм була використана бібліотека Qt Charts. Перш за все ми отримуємо всі файли в поточному каталозі за допомогою `QFileInfoList QDir::entryInfoList(QDir::Filters filters);`
Після цього ми переглядаємо цей список, визначаємо тип кожного файлу, а потім збільшуємо лічильник відповідного запису в `QMap <QString, int>`. Маючи цей розподіл, можна легко створити `QChart`.

```
1 QPieSeries *series = new QPieSeries();
2 QMapIterator<QString, int> it(typesDistributionMap);
3 while (it.hasNext()) {
4     it.next();
5     series->append(it.key(), it.value());
6 }
7 QChart typesChart = new QChart();
8 typesChart->addSeries(series);
```

3.2. UI/UX дизайн

3.2.1. Загальний огляд

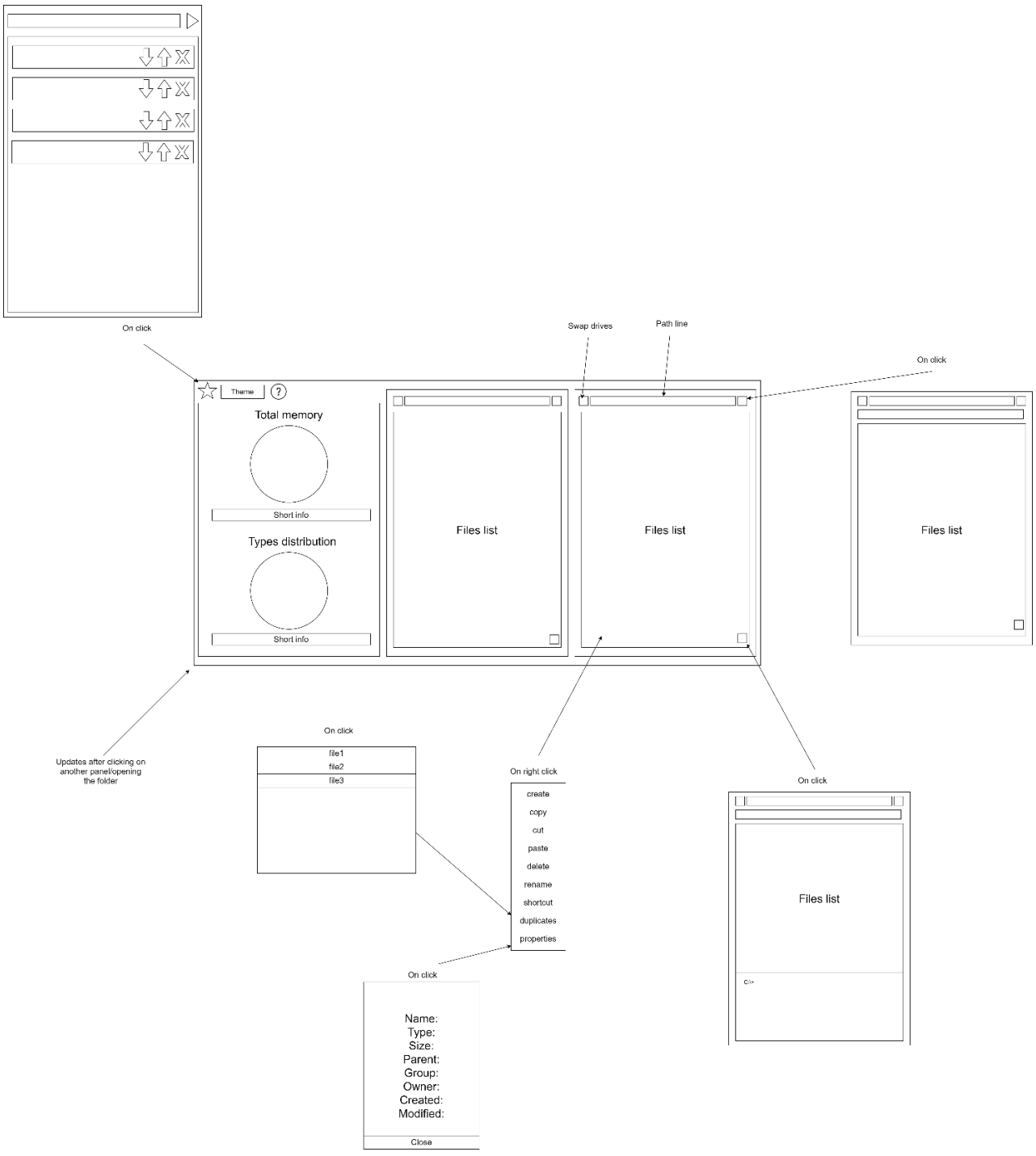

Рисунок 3.5. Перший прототип додатку.

На рисунку вище показаний перший прототип запланованої програми. Багато деталей було збережено в остаточному варіанті, хоча, звичайно, деякі були переглянуті, деякі видалені, а деякі речі додані. Наприклад, змінився набір кнопок для управління улюбленими шляхами, інформаційна панель змістилася вправо, до неї додалися нові віджети, наприклад хмарне сховище. Крім того, трохи змінився загальний вигляд програми. Тепер давайте подивимося на остаточний варіант дизайну.

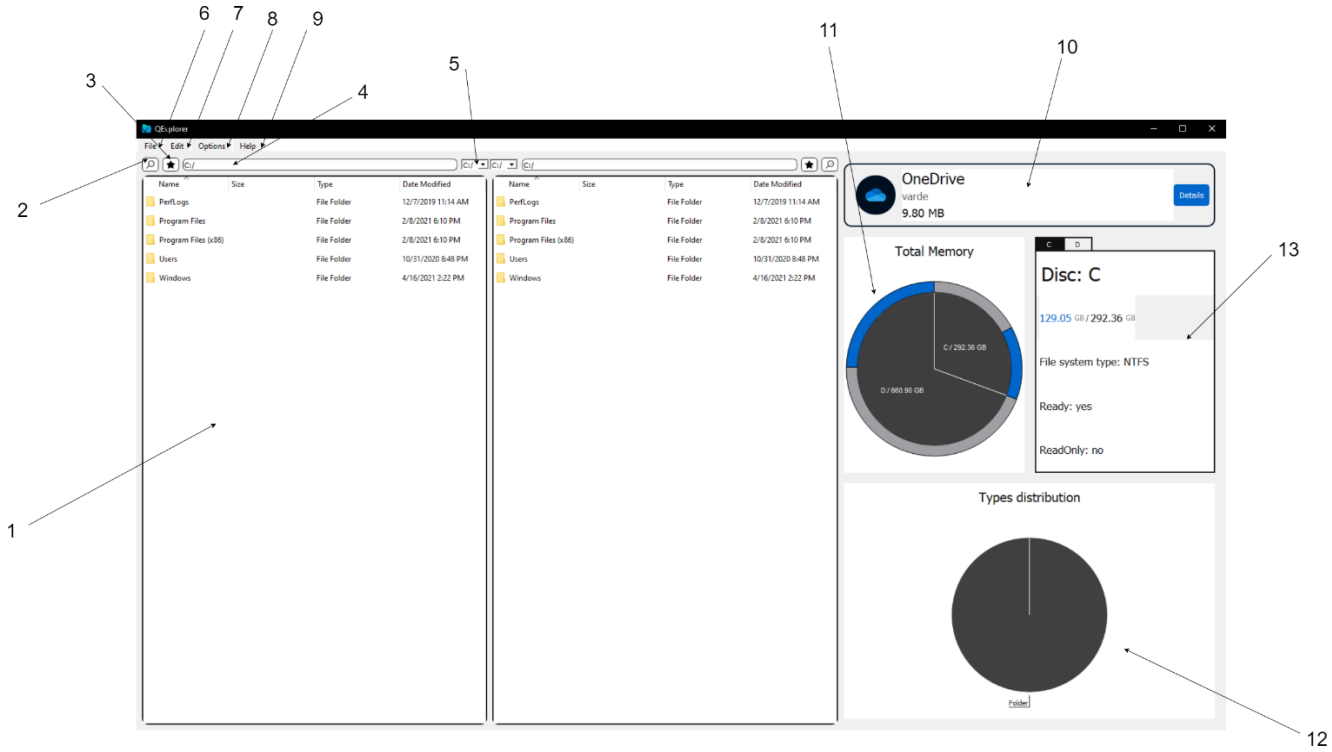

Рисунок 3.6. Остаточний прототип програми.

1. Список файлів

Цей список є місцем, де відображаються всі файли та папки. Елементи розташовано один за одним у алфавітному порядку за замовчуванням. Для кожного елемента відображається певний набір додаткової інформації. Це значок, призначений файлу в операційній системі, назва, тип, розмір і дата останньої модифікації. Ви можете використовувати розділ заголовка для сортування за кожним

зі стовпців. Докладну інформацію про сортування див. 3.3. Користувач може взаємодіяти зі списком файлів різними способами. Основне призначення – навігація. Ми можемо переходити з однієї папки в іншу, таким чином переміщаючись по файловій системі. Якщо ви двічі клацнете файл, він буде відкритий стандартною програмою, призначеною для нього. Для деяких додаткових операцій потрібно викликати контекстне меню. Це можна зробити, клацнувши правою кнопкою миші в будь-якому місці списку. Залежно від місця натискання відкриється меню з певним набором дій, показаних на малюнку нижче. Для отримання детальної інформації про файлові операції див. 3.1.

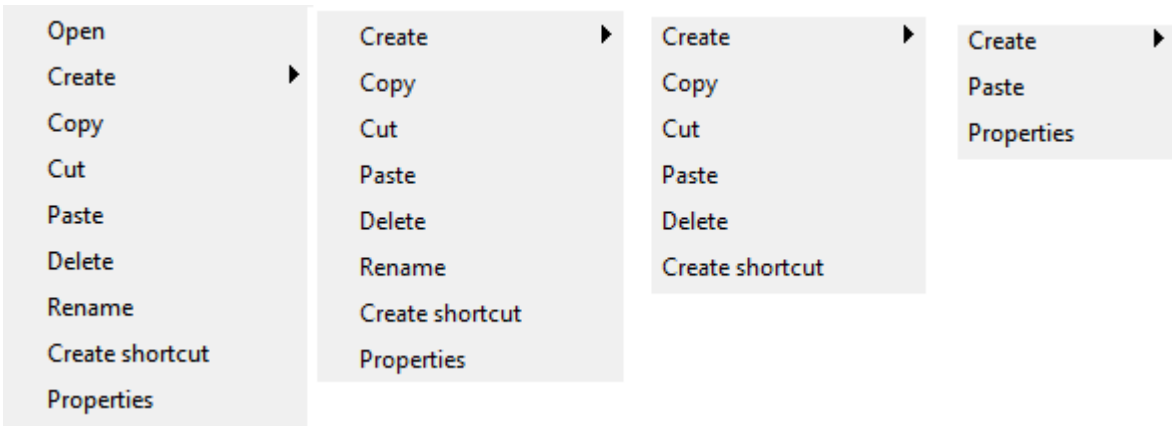

Рисунок 3.7. Контекстне меню при виділенні одного файлу, однієї папки, кількох об’єктів, та порожнього місця.

Також підтримується мультивибір. Це дозволяє вибрати кілька елементів і виконати операцію для всієї групи.

Кнопка для відкриття пошукового фільтра.

Після натискання цієї кнопки з'являється фільтр пошуку. Щоб приховати його, потрібно ще раз натиснути цю кнопку.

Кнопка для відкриття вікна прикріплених папок.

Після натискання цієї кнопки з’являється вікно прикріплених папок. Воно містить поле введення для додавання нового шляху, кнопку для підтвердження додавання та контейнер для наявних елементів

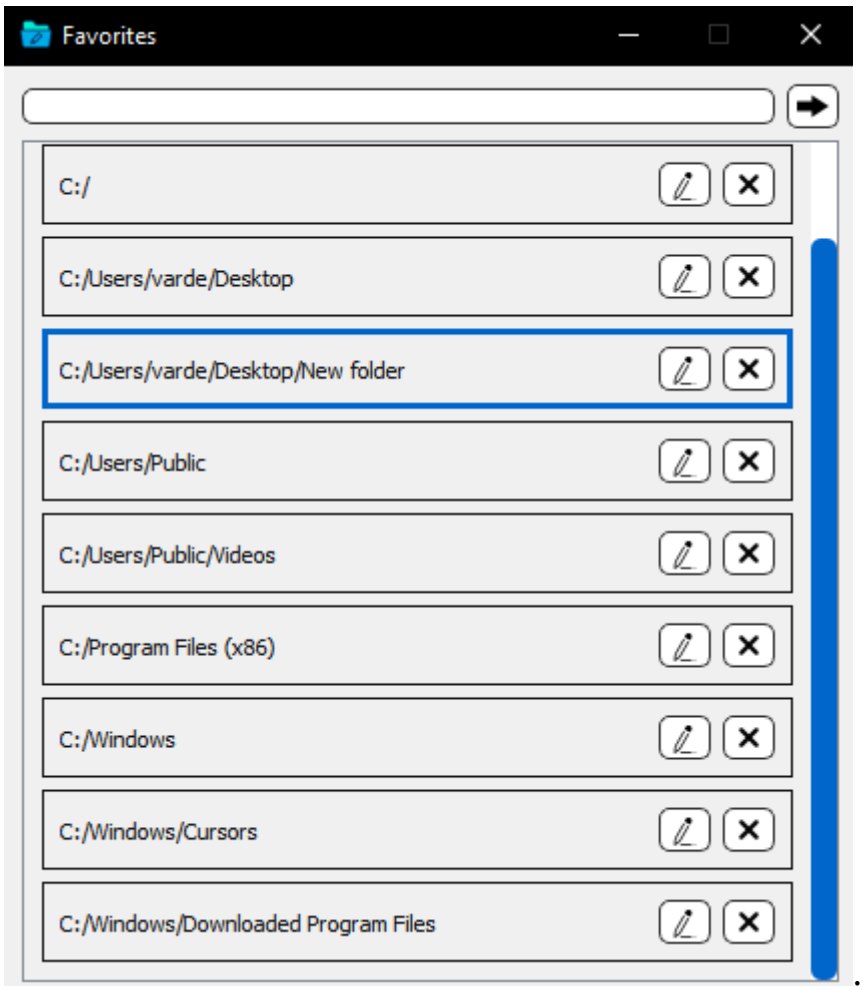

Рисунок 3.8. Вікно прикріплених папок.

За замовчуванням у рядку введення вводиться поточний шлях, що економить багато часу користувача, оскільки зазвичай, якщо хтось хоче додати папку до вибраного, він знаходиться в ній в даний момент. Після введення шляху потрібно натиснути кнопку праворуч або просто Enter на клавіатурі. Якщо шлях дійсний, він з’явиться в списку, після чого з ним можна буде взаємодіяти трьома способами. Перший варіант — перейти до каталогу, клацнувши лівою кнопкою миші на відповідному елементі. Другий — видалити прикріплену папку, натиснувши кнопку хрестика. Третій — відредагувати його, після чого буде проведено ще одну

перевірку. Щоб отримати докладну інформацію про функцію прикріплених папок, див. 3.6.

Рядок навігації

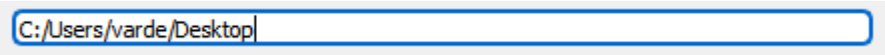

Рисунок 3.9. Рядок навігації.

Цей рядок представляє шлях у файловій системі, де зараз знаходиться користувач. Таким чином, кожен раз, коли ви переходите до каталогу, шлях, записаний у цьому полі, буде змінюватися. Це полегшить навігацію користувача. Крім того, ви можете прописати якийсь шлях у навігаційному рядку і, якщо він дійсний, відбудеться перехід до відповідного каталогу. Кожна панель має свій рядок. Для отримання детальної інформації про навігацію див. 3.4.

Комбінований список для заміни дисків

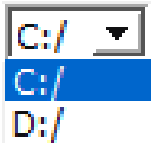

Рисунок 3.10. Комбінований список для заміни дисків.

Після натискання з’являється спливаюче вікно з усіма дисками. Доступно лише для Windows. Для отримання детальної інформації про навігацію див. 3.4.

Випадаюче меню «File»

Ще не використовується, додано для опцій, які з’являться в майбутньому.

Випадаюче меню «Edit»

Наразі спадне меню «Редагувати» містить опцію виклику глобального пошуку.

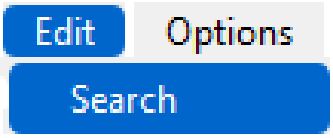

Рисунок 3.11. Випадаюче меню «Edit».

Вікно пошуку містить три поля для введення імені файлу, що містить текст і цільовий каталог. Після введення даних користувач повинен натиснути кнопку пошуку (або просто Enter на клавіатурі), і всі знайдені файли будуть відображені, як показано на малюнку нижче. Щоб отримати докладну інформацію про глобальний пошук, див 3.5.

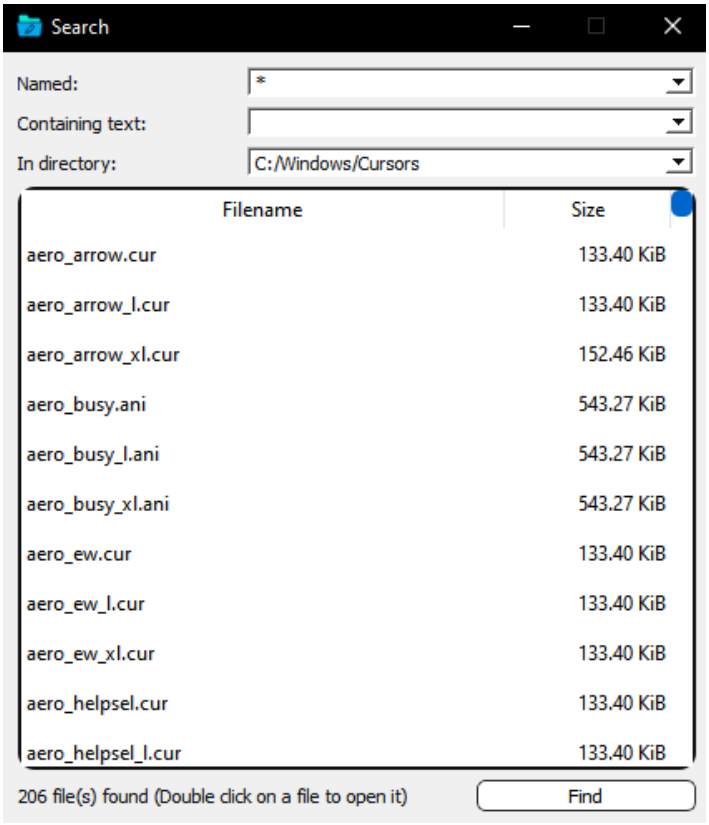

Рисунок 3.12. Вікно з результатами пошуку.

Випадаюче меню «Options»

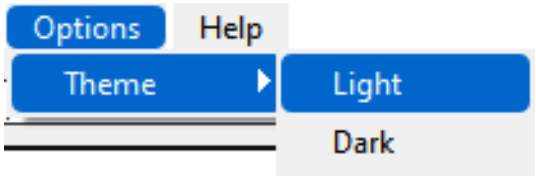

Рисунок 3.13. Випадаюче меню «Options».

Наразі, в меню «Options» можна змінити тему програми.

Випадаюче меню «Help»

Ще не використовується, додано для опцій, які з’являться в майбутньому.

Віджет хмарного сховища

Цей віджет відображає різну інформацію про хмарне сховище. Після натискання кнопки «Деталі» буде показано розподіл типів файлів на диску. Щоб закрити цей режим відображення, слід натиснути кнопку «Закрити». Докладну інформацію про підтримку хмарних дисків див. 3.7.

Рисунок 3.14. Два режими віджета хмарного сховища.

Діаграма загальної пам’яті

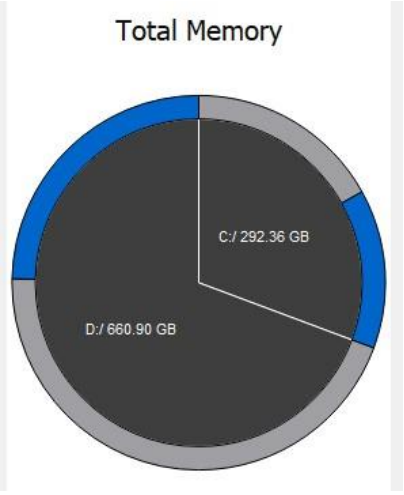

Рисунок 3.15. Діаграма загальної пам’яті.

На діаграмі вище показано розподіл загальної пам’яті пристрою. Це секторна діаграма з кількома фрагментами залежно від кількості дисків. Для кожного з цих фрагментів є дві зовнішні секції: використана пам’ять і вільна пам’ять. Таким чином, можна легко визначити, наскільки завантажений кожен диск. Щоб отримати докладну інформацію про інформаційну панель, див. 3.8.

Діаграма розподілу типів

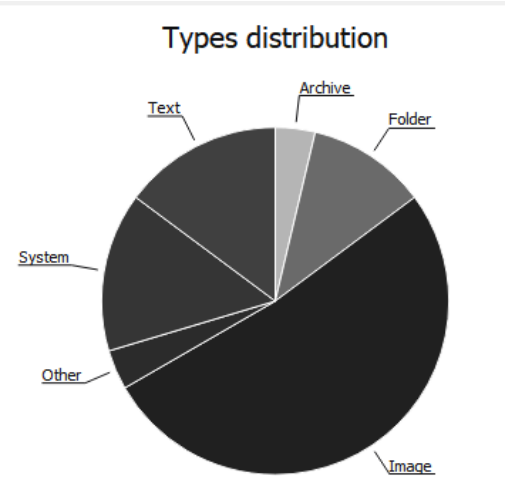

Рисунок 3.16. Діаграма розподілу типів.

На діаграмі показано розподіл типів у папці. Оновлюється, коли ви переходите до іншого каталогу. Для полегшення візуального сприйняття великої кількості типів зовні зрізів показані мітки. Щоб отримати докладну інформацію про інформаційну панель, див. 3.8.

Віджет з інформацією про диски

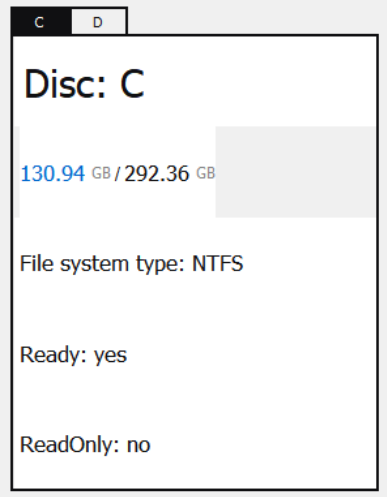

Рисунок 3.17. Віджет з інформацією про диски.

Віджет відображає деяку інформацію про всі наявні диски у файловій системі. Користувач може переключатися між дисками за допомогою кнопок вище. Кнопка

активної вкладки змінює колір, щоб було інтуїтивно зрозуміло, на якій із них зараз перебуває користувач. Щоб отримати докладну інформацію про інформаційну панель, див. 3.8.

3.2.2. **Користувацький** потік

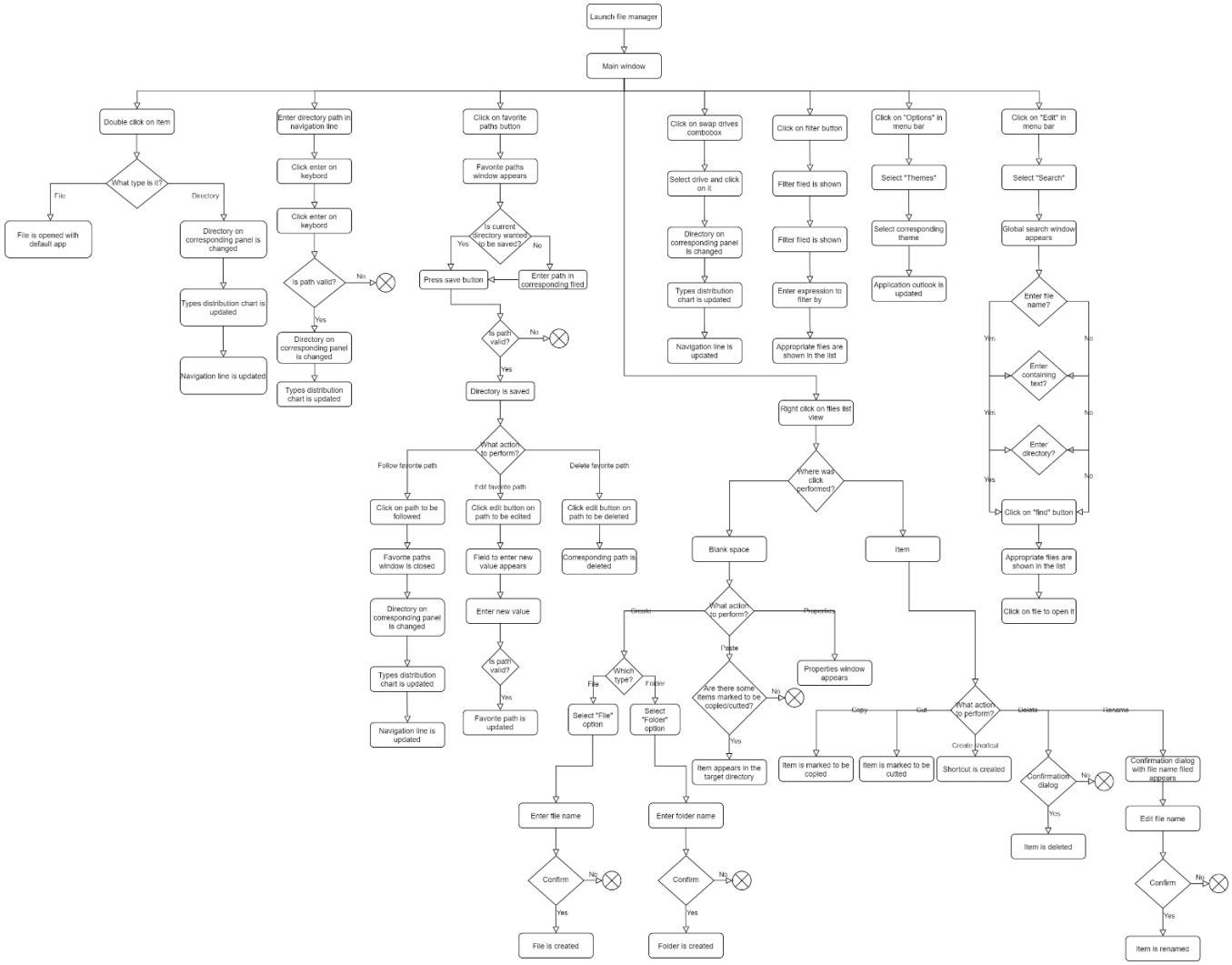

Рисунок 3.18.Користувацький потік.

3.2.3. Теми

Останнім часом стала популярною тенденція додавати темну тему в додатки та сайти. На цю тему вже з'явилося багато досліджень.

Хоча темна тема може надати низку переваг деяким користувачам із вадами зору, особливо тим, хто має катаракту, результати досліджень вказують на перевагу позитивної полярності для користувачів із нормальним зором. Іншими словами, для користувачів із нормальним зором світла тема в більшості випадків призведе до кращої продуктивності.

Ці результати найкраще пояснюються тим фактом, що при позитивній полярності контрасту є більше світла і, отже, більше звуження зіниці. Результатом є менше сферичних аберацій, більша глибина різкості та загальна краща здатність фокусуватися на деталях без напруження очей.

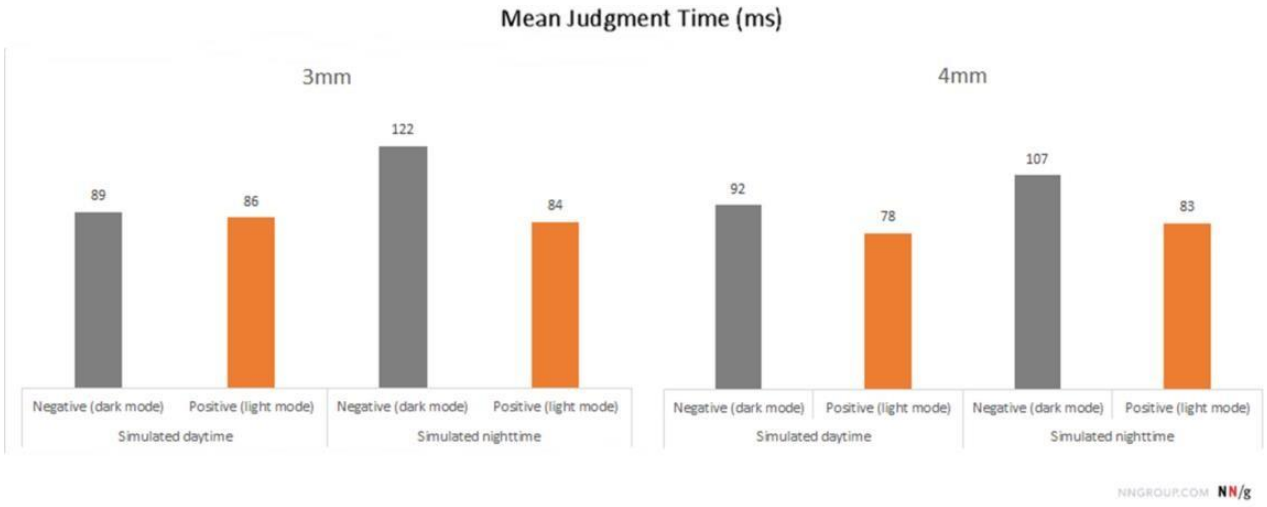

Рисунок 3.19. Середній час фокусування.

У той же час, необхідно дозволити користувачам перемикатися на темну тему за бажанням – з трьох причин: можуть бути довгострокові ефекти, зі світлою темою; деяким людям із вадами зору краще працювати на темній темі; а деяким користувачам просто подобається темна тема. (Ми знаємо, що люди рідко змінюють налаштування за замовчуванням, але вони повинні мати можливість це зробити.)

Малоймовірно, що люди змінять режим відображення для випадкового сайту, але якщо вони використовують веб-сайт або програму, їм це може знадобитися. Зокрема, додатки, призначені для читання в електронних формах (таких як книги, журнали та навіть сайти новин), повинні пропонувати темну тему. І в ідеалі ця опція має застосовуватися до всіх сторінок цього сайту чи програми.

1) Світла тема

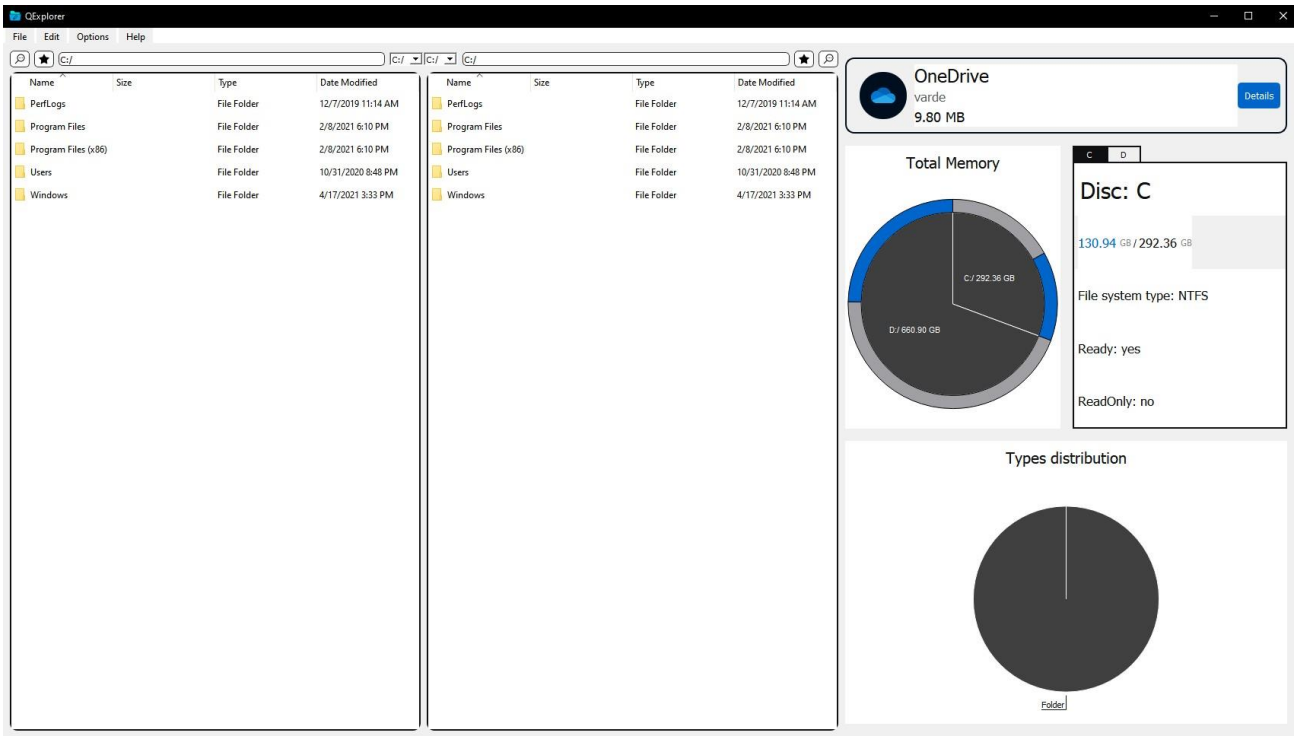

Рисунок 3.20. Світла тема.

Основним кольором у цій темі є білий(#FFFFFF). Він покриває близько 85% площі застосування. Усі інші елементи, такі як символи, рамки, піктограми тощо, переважно чорного кольору(#000000). Коли ви активуєте деякі елементи, вони змінюють свій колір, щоб візуально підкреслити дію користувача. Цей колір активації — королівський синій(#0066CC). В інтерфейсі багато рамок. Це робиться для того, щоб виділити елементи і дати користувачеві знати, де він може з ними взаємодіяти, а де ні. Це можна зробити за допомогою кольору фону, як це відбувається у випадку темної теми, але для світлої було вирішено залишити цей колір незмінним для більшості елементів, тому рамки є найкращим вибором.

2) Темна тема

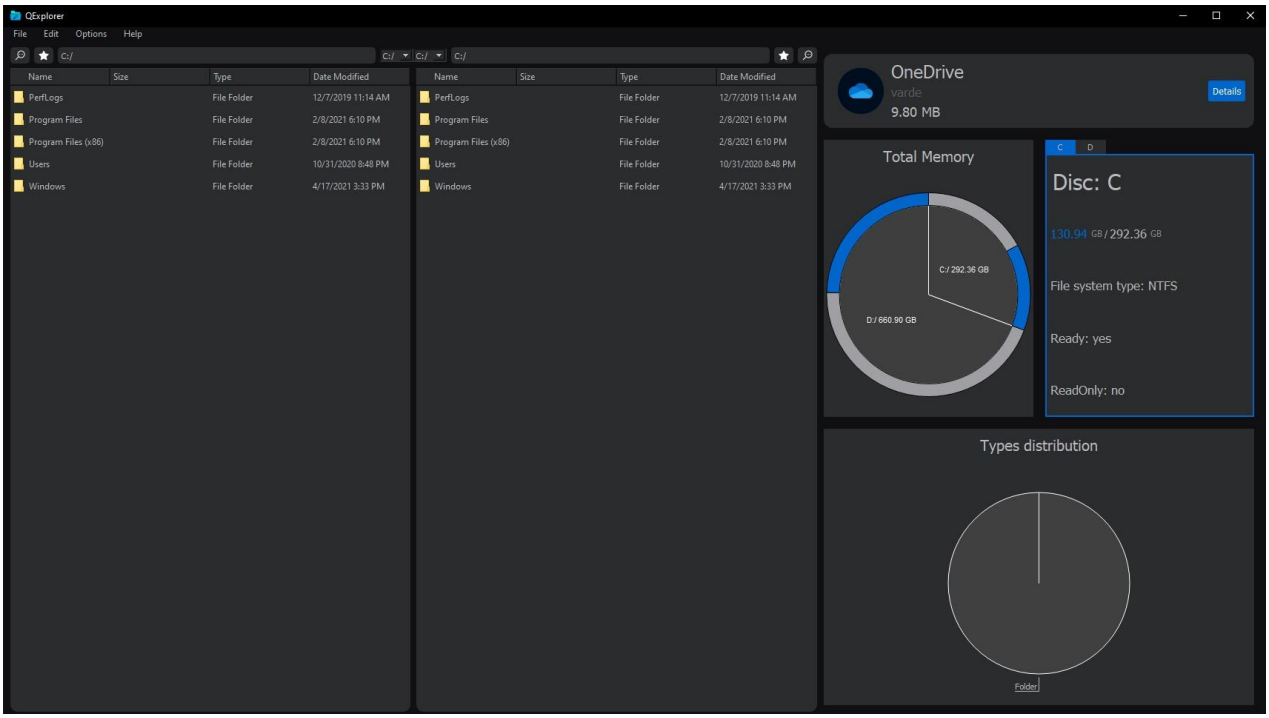

Рисунок 3.21. Темна тема.

Основний колір, який використовується в цій темі, — темно-сірий(#2B2C2D). Він служить фоновим кольором для всіх основних елементів інтерфейсу. Для того, щоб можна було розрізняти елементи, фоновий колір вікна був встановлений на чорний(#101012). Для кольору шрифту та іконок був обраний сріблястий(#BABABF). Це було зроблено для того, щоб основний текст відповідав стандарту WCAG AA принаймні 4,5:1 при нанесенні на поверхні з найвищим (і найменшим) рівнем яскравості. У нашому випадку коефіцієнт контрастності дорівнює 7,23, що є досить оптимальним. Як і у випадку зі світлою темою, існує колір активації, який також є королівським синім(#0066CC).

3.3. Оцінка

3.3.1. Тестове середовище

ОС: Windows/Ubuntu

ЦП: кількість ядер ЦП: 8, кількість потоків: 16, базова тактова частота: 3,6 ГГц, кеш L1: 512 КБ, кеш L2: 4 МБ, кеш L3: 32 МБ

Материнська плата: чіпсет: AMD X570, форм-фактор: ATX, підтримка ЦП: AMD Socket AM4, підтримка пам'яті: двоканальна

Оперативна пам’ять: тип пам’яті: DDR4, об’єм: 16 ГБ (8 ГБ x 2), перевірена швидкість: 3200 МГц, перевірена затримка: 16-18-18-38, швидкість SPD: 2133 МГц

SSD: ємність: 1 ТБ, швидкість читання: до: 550 МБ, швидкість запису: до: 520 МБ, тип комірки: TLC

3.3.2. Результати

Таблиця 3.3

Metric	Value	
	Windows	Ubuntu
Min CPU Usage	0.10%	0%
Max CPU Usage	13.70%	17%
Min RAM Usage	22.0 MB	25.4 MB
Max RAM Usage	148.0 MB	117.2 MB
Launch time	0.53 sec	0.58 sec
Application size	17.4 MB	

Таблиця 3.4

Назва Класу	**Стрічки**	**Методи**
MainWindow	575	28
SearchWindow	179	11
StatisticsWidget	129	5
Properties	98	17

CreationUtils	89	5
ConfigParser	82	5
StatisticsUtils	78	4
CloudDriveWidget	73	6
DonutBreakdownChart	73	4
DirectorySizeCalculationUtils	66	5
CopyPasteUtils	56	3
RenameUtils	48	3
DeletionUtils	46	3
CloudDriveUtils	42	3
CloudDrive	41	8
FavoritePathsContainer	34	3
MainSlice	30	5
DiscView	22	2
SwapDrivesUtils	22	2
CopyPathUtils	19	1
FavoritesMainWindow	17	3
NavigationUtils	15	1
FiltersUtils	15	1
PropertiesWindow	12	2

Таблиця 3.5

Розширення	**Призначення**	**К-сть стрічок**
.cpp	Основна бізнес-логіка та визначення поведінки програми	1956
.h	Оголошення різних конструкцій, таких як класи, перелічувальні	774

	константи тощо.	
.css	Визначення кольорів, шрифтів, стилів, розташування окремих блоків та інших аспектів подання зовнішнього вигляду додатку	558
.ui	Визначення графічних компонентів програми	811
Загалом		4099

РОЗДІЛ 4. РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ

4.1. Опис ідеї проекту

Завдання проекту полягає в створенні швидкого, зручного файлового менеджера, який би покривав недоліки наявні, в багатьох інших аналогах, та об'єднював в собі весь корисний функціонал. Додаток буде призначений для особистого щоденного користування, при роботі з файлами, папками та файловою системою загалом. Основними перевагами, які отримують користувачі є:

- Швидкодія

Детальніше про це в 5 розділі. Програма має швидкий запуск, та споживає мінімум системних ресурсів, що дозволяє використовувати її будь-де, без обмежень.

- Адаптивність

Користувацький інтерфейс при запуску може адаптуватись до всіх розширень та екранів.

- Зручний UI/UX дизайн

При розробці проекту велику увагу приділено саме цьому аспекту. Для обох тем було підібрано оптимальну кольорову гаму. Користувацький потік було зроблено максимально горизонтальним, де це можливо. Для всіх кнопок підібрані інтуїтивно зрозумілі значки або написи.

- Оптимальний функціонал

Необхідний функціонал був визначений на основі попередньо проведеного дослідження існуючих на ринку аналогів. В пріоритеті були ті функції які необхідні для сучасного файлового менеджера, та водночас не будуть шкодити виконанню всіх інших вимог, перечислених в даному списку.

- Працездатність на різних ОС

Роботу додатка протестовано на трьох операційних системах: Windows, Ubuntu та macOS. Це зроблено для того щоб охопити максимальне коло користувачів.

4.2. Існуючі рішення

4.2.1. Total Commander

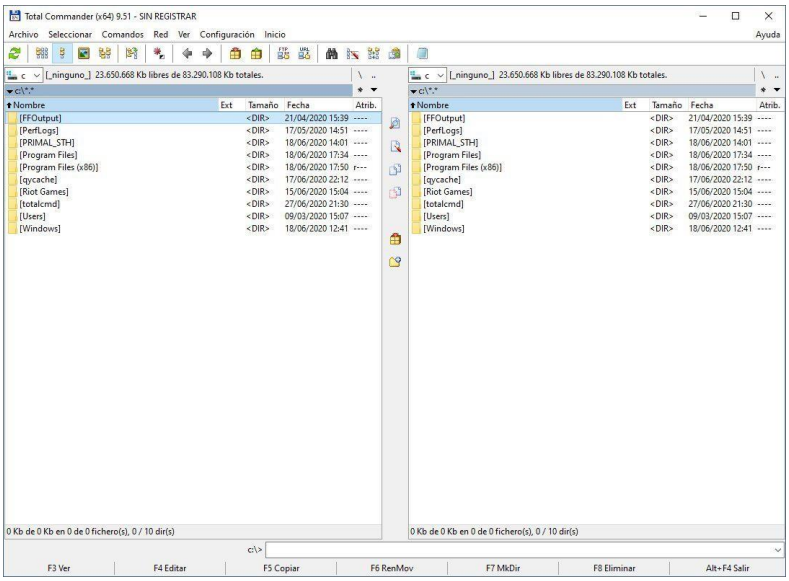

Рисунок 4.1. Total Commander.

Total Commander – це двопанельний файловий менеджер для 32- та 64-розрядних операційних систем Windows, який дозволяє не тільки спростити рутинні операції з файлами (копіювання, вставка, видалення, переміщення тощо), але й має багато додаткових корисних опцій.

Функціонал:

- Показати/вибрати файли з певним шаблоном пошуку, розміром, датою або вмістом.
- Розширена функція пошуку з повнотекстовим пошуком у будь-яких файлах на кількох дисках.
- Командний рядок для запуску програм з параметрами.
- Розділяти/об’єднувати великі файли.
- Перегляд користувацьких стовпців для відображення додаткових відомостей про файл.
- Порівняти файли за вмістом.
- Пошук дублікатів файлів.

- Вбудований FTP-клієнт підтримує більшість публічних FTP-серверів і деякі мейнфрейми.
- Настроюване головне меню.
- Робота з архівами.
- Функція передачі через паралельний порт (пряме кабельне підключення).

Таблиця 4.1

Переваги	**Недоліки**
Підтримка плагінів	Необхідно витратити багато часу на індивідуальне налаштування
Дуже стійкий	Важко звикнути
Дозволяє працювати без миші	
Можливість налаштування	
Багатофункціональний	

4.2.2. Directory Opus

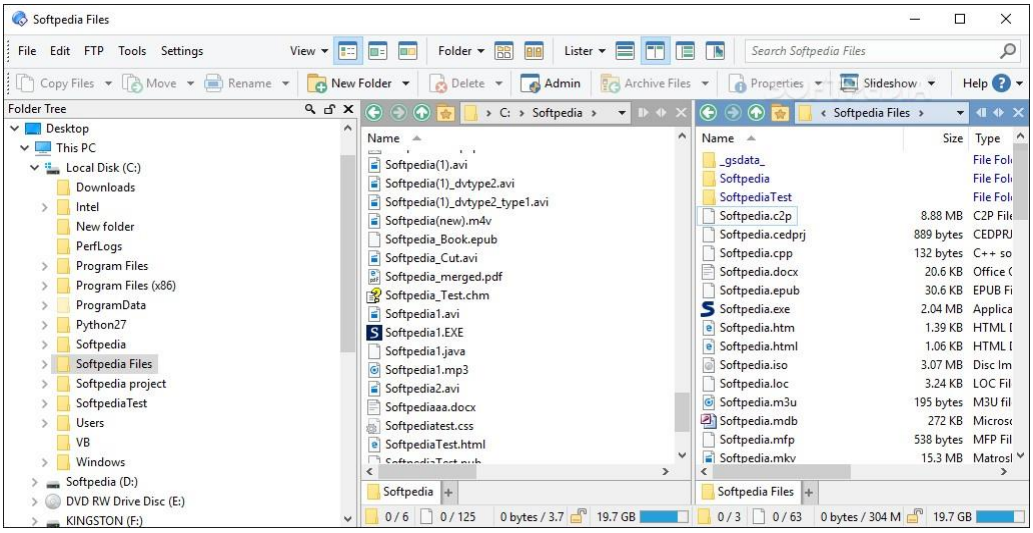

Рисунок 4.2. Directory Opus.

Directory Opus – це потужний і простий у використанні файловий менеджер, який забезпечує всі основні операції з файлами. Можливості менеджера не відрізняються чимось особливим в порівнянні з аналогічними програмами. Розробники відзначають, що Directory Opus задумувався в основному як заміна стандартного Windows Explorer. Таким чином, цей додаток полегшить роботу з

файлами, зробить її більш зручною та комфортною. Directory Opus підтримує роботу з архівами, надає доступ по FTP, обробляє музичні та графічні файли, синхронізується з хмарним сховищем OneDrive і багато іншого.

Функціонал:

- Вкладки дозволяють зберігати декілька відкритих папок і швидко перемикатися між ними.
- Інтегрована панель перегляду для попереднього перегляду багатьох поширених форматів файлів зображень і документів.
- Перегляд і редагування метаданих файлів (EXIF, MP3, PDF тощо).
- Сортування, групування, фільтрація та пошук.
- Підтримка FTP, Zip, 7-Zip, RAR і багатьох інших форматів архівів.
- Доступ до вмісту на портативних пристроях, таких як телефони, планшети та камери.
- Вбудовані інструменти, включаючи синхронізацію, пошук дублікатів файлів, конвертер зображень і завантажувач.
- Друк або експорт списків папок, копіювання списків файлів у буфер обміну, обчислення розмірів папок.
- Підтримка запису CD/DVD.
- Повністю настроюваний інтерфейс користувача - кольори, шрифти, панелі інструментів, гарячі клавіші клавіатури.
- Повний інтерфейс сценаріїв підтримує VBScript, JScript або будь-яку сумісну встановлену мову Active Scripting.

Таблиця 4.2

Переваги	**Недоліки**
Настроюваний інтерфейс користувача	Складний для недосвідчених користувачів
Вкладки	Займає багато пам'яті
Постійне обслуговування програми та підтримка клієнтів	Дорогий

4.2.3. Windows Explorer

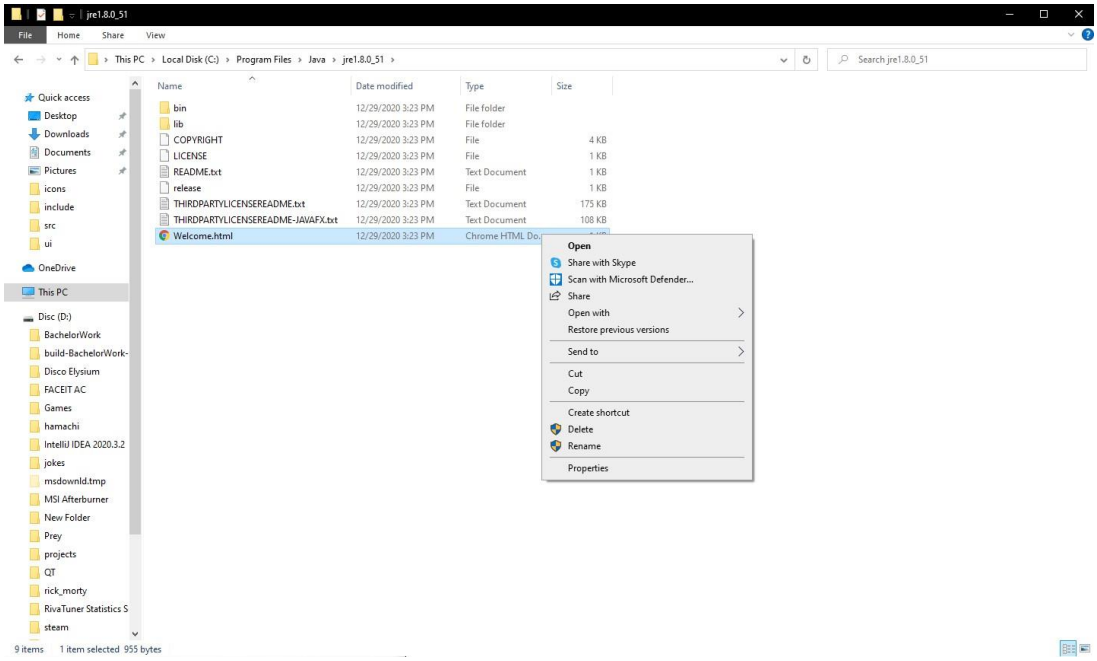

Рисунок 4.3. Windows Explorer.

Windows Explorer – це програма, яка реалізує графічний інтерфейс користувача для доступу до файлів в операційній системі Microsoft Windows. Провідник тепер є де-факто основою графічного інтерфейсу користувача для Windows. Все, що бачить користувач після завантаження Windows - значки на робочому столі, панель завдань, меню «Пуск» - це Провідник Windows.

Функціонал:

- Доступ до часто використовуваних команд.
- Миттєвий пошук, який показує лише ті файли, які відповідають тому, що ви ввели у полі пошуку для поточної папки та будь-якої з її вкладених папок.
- Відображення загальних папок, таких як «Вибране», «SkyDrive», «Домашня група» (спільна мережа), «Цей ПК» і «Мережа».
- Розширена інформація про файл (метадані).
- Настроюваний інтерфейс.
- Підтримка хмарного сховища.

- Очищення дисків.
- Оптимізація та дефрагментація дисків.

Таблиця 4.3

Переваги	Недоліки
Попередньо встановлено на Windows	Надто спрощений
Гарний вибір для користувачів із низькими вимогами	Малий функціонал
	Неможливо налаштувати

4.2.4. Q-Dir

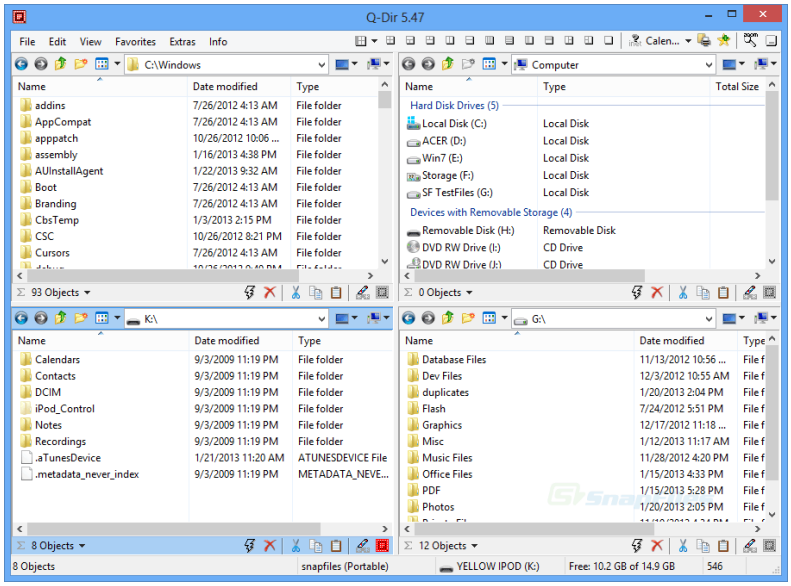

Рисунок 4.4. Q-Dir.

Q-Dir – це файловий менеджер із чотирьохпанельним інтерфейсом, підтримкою zip-архівів, FTP і виділенням папок і файлів із різними розширеннями. За замовчуванням Q-Dir використовує інтерфейс із чотирма панелями (4 вікна), але надає можливість налаштувати інтерфейс, дозволяючи вказати 3 або 2 панелі з вертикальним або горизонтальним розташуванням. Q-Dir тісно інтегрований із Windows Explorer. Наприклад, використовуються стандартні види панелей. Списки, таблиці, мініатюри - все взято зі стандартного інструменту Windows.

Функціонал:

- Управління файлами в 4 вікнах з вкладками.
- Розмір папки з додатковою інформацією.
- Кольоровий фільтр для файлів і папок.
- Структура каталогу з видимими гілками дерева.
- Створено на основі файлового менеджера MS Windows.
- Повна підтримка Unicode.
- Покращено попередній перегляд файлів Quad Explorer.
- Багатомовний.
- Низьке використання системних ресурсів.

Таблиця 4.4

Переваги	**Недоліки**
Низькі вимоги системних ресурсів	Присутні баги
Вкладки	Деякі критичні функції, як наприклад пошук, відсутні
Можливість налаштування	
Безкоштовний	

4.2.5. Nautilus

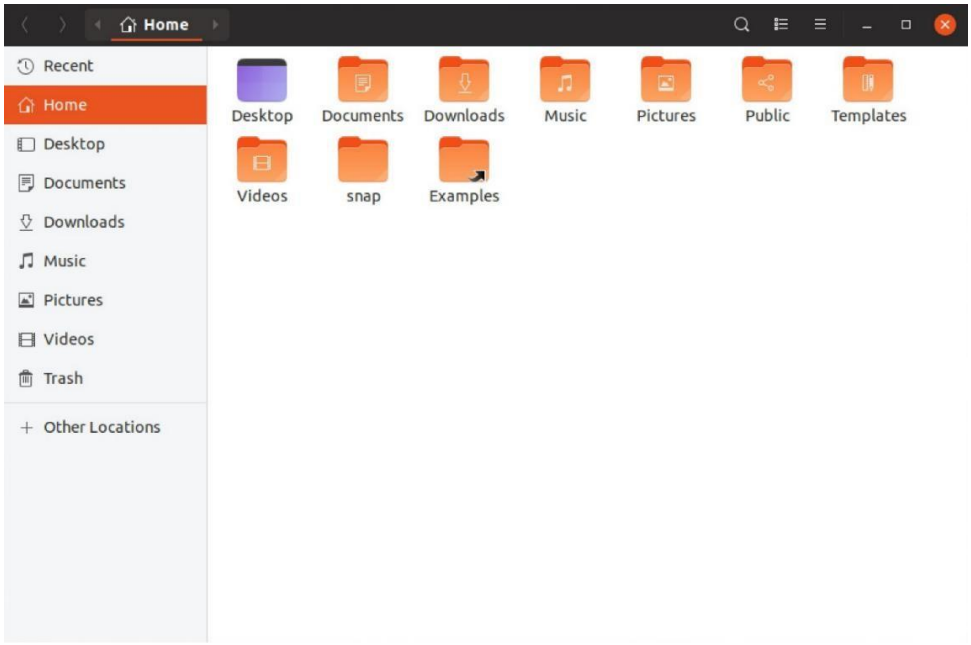

Рисунок 4.5. Nautilus.

Nautilus – це файловий менеджер для робочих середовищ GNOME та Unity. В Ubuntu він встановлений як головний файловий менеджер, а також як менеджер робочого столу.

Функціонал:

- Підключення та відключення накопичувачів (жорсткі диски, мережеві накопичувачі, флешки, оптичні приводи тощо).
- Робота з віддаленими серверами (FTP, SSH, WebDAV, SMB).
- Переглядайте мініатюри файлів (відео, зображення, PDF, DJVU, текстові файли).
- Перегляд властивостей файлів і каталогів (включаючи додаткові властивості на окремих вкладках за допомогою сторонніх програм).
- Створюйте, змінюйте, видаляйте файли та каталоги (включаючи використання шаблонів файлів, розташованих у каталозі ~ / Templates або ~ / Templates).
- Запуск скриптів і програм.
- Пошук файлів і каталогів за назвою.
- Запис CD / DVD дисків (за допомогою Brasero).

Таблиця 4.5

Переваги	**Недоліки**
Простий у використанні	Дуже мало функціональних можливостей
Мінімалістичний	Неможливо налаштувати
Підтримується на всіх дистрибутивах Linux	Витоки пам'яті
	Багато багів

Загальне порівняння

Таблиця 4.6

Файловий менеджер	Total Commander	Directory Opus	Windows Explorer	Q-Dir	Nautilus	Розроблений додаток
Підтримувана ОС	Windows, Windows CE, Android	AmigaOS, Windows	Windows	Windows	Linux	Windows, Linux, macOS
Пошук файлів за допомогою регулярних виразів	+	+	-	+	+	+
FTP клієнт	+	+	+	-	+	-
Пошук файлів за вмістом	+	+	-	-	-	+
Мультивибір	+	+	+	+	+	+
Робота з архівами	-	+	+	+	+	-
Настроюваний інтерфейс користувача	+	+	+	+	-	+
Збереження обраних папок	+	-	+	+	-	+
Багатомовний	+	+	+	+	+	-
Підтримка хмарного сховища	-	-	+	-	-	+
Панель системної інформації	-	-	-	-	-	+
Глобальний пошук файлів	+	+	+	+	+	+
Кілька робочих панелей	+	+	-	+	-	+
Вбудований командний рядок	+	-	-	-	-	-

ВИСНОВКИ

У цій роботі створено швидкий, зручний і простий у використанні файловий менеджер, який відповідає всім потребам сучасного користувача. У розробці було введено деякі нові функції, які нечасто зустрічаються в цій області, наприклад інформаційна панель, закріплені каталоги, глобальний пошук за вмістом файлів, кілька тем тощо. На основі проведених тестувань було виявлено, що даний файловий менеджер використовує приблизно в 3 рази менше системних ресурсів. Працездатність перевірялась на 3 найпопулярніших операційних системах: Windows 10, Ubuntu 22.04.1, macOS 13.

Робота,яку потрібно зробтити:

1) FTP клієнт

Програма для легкого доступу до FTP-сервера. Таким чином файловий менеджер стає FTP-клієнтом, через який можна отримувати файли через мережу. Функція ще не реалізована, через те, що в останній версії Qt прибрала підтримку FTP, і перехід на попередню версію вимагає багато часу для адаптації коду.

2) Вбудована командна стрічка

Інтерфейс командного рядка (CLI) для взаємодії з операційною системою шляхом введення різних команд. Ця функція вже була в проекті, але її довелося видалити через велику кількість помилок і кількість часу, необхідного для їх усунення. Основна проблема полягає в тому, що проект є кросплатформенним, і API суттєво відрізняється на різних операційних системах.

3) Drag & drop

Поки що підтримується лише зовнішнє перетягування. Це означає, що ми можемо перетягувати файли в інші програми, але ми не можемо робити це всередині файлового менеджера.

Відомі баги:

● Назви стовпців змінюють шрифт з жирного на звичайні після натискання.

● Хмарні диски не знайдені на деяких ПК.

● Частини діаграм не видно при деяких роздільних здатностях екрана.

● за певних умов повзунок покриває важливу частину інтерфейсу.

● Проблеми стилю: неправильні межі, зайві поля та відступи, порушення колірної теми в деяких місцях тощо.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Таненбаум Е., Бос Х. Сучасні операційні системи. 4-е вид. СПб, 2015, 1120 с.

2. Шеховцов В.А. Операційні системи. ТОВ «Видавнича група BHV», 2005, 576 с.

3. Шлеє. М. Qt 5.10. Професійне програмування на C++. БХВ-Петербург, 2016, 1072 с.

4. Купер А., Рейманн Роберт М. Інтерфейс. Основи проектування взаємодії. Пітер Прес, 2021, 720 с.

5. Роберт Лав. Linux. Системне програмування. 2-е вид. Пітер Прес, 2014, 448 с.

6. Марк Руссинович, Внутрішнє улаштування Windows. 6-те вид. Пітер Прес, 2013, 800 с.

7. Steve D. Pate. UNIX Filesystems: Evolution, Design, and Implementation. 1st Edition. Wiley, 2003, 472 p.

8. Jasmin Blanchette, Mark Summerfield. C++ GUI Programming with Qt 4. 2nd Edition, Prentice Hall, 2008, 752 p.

9. Guillaume Lazar, Robin Penea. Mastering Qt 5. Second Edition, Packt Publishing, 2018, 376 p.

10. New Trends in GUI. SendPoints Sendpoints Publishing Company Limited, 2018, 256 p.

11. Волянський В.С., Борецька І.Б. Розроблення кросплатформного менеджера файлової системи для роботи з хмарними технологіями. Комп'ютерне моделювання та інформаційні технології: четверта науково-практична конференція студентів, аспірантів та молодих вчених. м. Львів: кафедра інформаційних технологій НЛТУ України, 2022 р., 77-81 с.

12. Офіційна документація Qt5 https://doc.qt.io/qt-5

13. Хмарні сховища https://en.wikipedia.org/wiki/Cloud_storage

ДОДАТКИ

cloud_drives.cpp

```
const QStringList CloudDriveUtils::supported_drives = {"/OneDrive"};

QList<CloudDrive> CloudDriveUtils::get_supported_drives()
{
    QList<CloudDrive> drives;
    for (auto drive : supported_drives) {
        QString path = QDir::homePath().append(drive);
        if (QDir(path).exists()) {
            drives.append(get_drive(path));
        }
    }
    return drives;
}

CloudDrive CloudDriveUtils::get_drive(QString path)
{
    QFileInfo driveInfo(path);
    CloudDrive drive;
    QFileIconProvider iconProvider;
    drive.setName(driveInfo.baseName());
    drive.setPath(path);
    drive.setImage(iconProvider.icon(driveInfo).pixmap(QSize(32, 32)));
    drive.setSize(get_drive_size(path));
    return drive;
}

qint64 CloudDriveUtils::get_drive_size(QString path)
{
    qint64 size = 0;
    QDir dir(path);
    QDir::Filters fileFilters = QDir::Files | QDir::System | QDir::Hidden;
    for (QString filePath : dir.entryList(fileFilters)) {
        QFileInfo fi(dir, filePath);
        size += fi.size();
    }
    QDir::Filters dirFilters = QDir::Dirs | QDir::NoDotAndDotDot | QDir::System | QDir::Hidden;
    for (QString childDirPath : dir.entryList(dirFilters))
        size += get_drive_size(path + QDir::separator() + childDirPath);
    return size;
}
```

config_parser.cpp

```
#include "./include/utils/config_parser.h"

const std::string ConfigParser::CONFIG_LOCATION = "./config/.config";
const std::string ConfigParser::TEMP_CONFIG_LOCATION = "./config/.config.temp";


void ConfigParser::configure()
{
    parse_config();
}
```

```
void ConfigParser::parse_config()
{
    std::string line;
    std::fstream fileStream(CONFIG_LOCATION);
    while (std::getline(fileStream, line)) {
        std::istringstream is_line(line);
        std::string key;
        if (std::getline(is_line, key, '=')) {
            std::string value;
            if (key[0] == '#')
                continue;
            if (std::getline(is_line, value)) {
                if (key == "theme") {
                    if (value == "LIGHT") {
                        currentTheme = Theme::LIGHT;
                    } else if (value == "DARK") {
                        currentTheme = Theme::DARK;
                    } else {
                        currentTheme = Theme::DARK;
                    }
                }
                if (key == "favorites") {
                    for (QString path : QString(value.c_str()).split(",")) {
                        existingFavoritePaths.append(path);
                    }
                }
            }
        }
    }
    fileStream.close();
}

void ConfigParser::change_config(QString key, QString value)
{
    std::ifstream in(CONFIG_LOCATION);
    std::ofstream out(TEMP_CONFIG_LOCATION);
    std::string strTemp;
    const char *delim = "=";
    while (std::getline(in, strTemp)) {
        if (split(strTemp, delim)[0] == key.toStdString()) {
            strTemp = key.append(delim).append(value).toStdString();
        }
        strTemp += "\n";
        out << strTemp;
    }
    in.close();
    out.close();
    std::remove(CONFIG_LOCATION.c_str());
    std::rename(TEMP_CONFIG_LOCATION.c_str(), CONFIG_LOCATION.c_str());
}

std::vector<std::string> ConfigParser::split(const std::string &str, const char *delim)
{
    std::vector<std::string> dest;
    char *pTempStr = strdup( str.c_str() );
    char *pWord = strtok(pTempStr, delim);
```

```
    while (pWord != NULL) {
        dest.push_back(pWord);
        pWord = strtok(NULL, delim);
    }
    free(pTempStr);
    return dest;
}

QString ConfigParser::list_to_string(QStringList list)
{
    QString result = "";
    for (QString string : list)
        result.append(string + ",");
    return result.remove(result.size() - 1, 1);
}
```

copy_path.cpp

```
#include "./include/utils/copy_path.h"


void CopyPathUtils::copyPath(QString src, QString dst)
{
    QDir dir(src);
    if (! dir.exists())
        return;

    foreach (QString d, dir.entryList(QDir::Dirs | QDir::NoDotAndDotDot)) {
        QString dst_path = dst + QDir::separator() + d;
        dir.mkpath(dst_path);
        copyPath(src + QDir::separator() + d, dst_path);
    }

    foreach (QString f, dir.entryList(QDir::Files)) {
        QFile::copy(src + QDir::separator() + f, dst + QDir::separator() + f);
    }
}
```

copypaste.cpp

```
#include <./include/utils/copypaste.h>

void CopyPasteUtils::paste_unit(QFileInfo copy_info, QString path, bool isFolder)
{
    if (isFolder) {
        paste_folder(copy_info, path);
    } else {
        paste_file(copy_info, path);
    }
}

void CopyPasteUtils::paste_file(QFileInfo copy_info, QString path)
{
    QFile file_to_copy(copy_info.absoluteFilePath());
    if (file_to_copy.exists()) {
        QString new_path = path;
        new_path.append("/");
```

```
        new_path.append(copy_info.baseName());
        if (copy_info.completeSuffix().size() != 0) {
            new_path.append(".");
            new_path.append(copy_info.completeSuffix());
        }
        int counter = 1;
        while (!file_to_copy.copy(new_path)) {
            new_path = path;
            new_path.append("/");
            QString text = copy_info.baseName();
            if (copy_info.completeSuffix().size() != 0) {
                text.append(".");
                text.append(copy_info.completeSuffix());
            }
            QStringList text_list = text.split('.');
            text_list[0].append("(");
            text_list[0].append(QString::number(counter));
            text_list[0].append(")");
            new_path.append(text_list.join("."));
            counter++;
        }
    }
}

void CopyPasteUtils::paste_folder(QFileInfo copy_info, QString path)
{
    QDir dir_to_copy(copy_info.absoluteFilePath());
    if (dir_to_copy.exists()) {
        QString new_path = path;
        QString src_path = copy_info.absoluteFilePath();
        new_path.append("/");
        new_path.append(copy_info.baseName());
        if (!copy_info.completeSuffix().isEmpty()) {
            new_path.append(".");
        }
        new_path.append(copy_info.completeSuffix());
        CopyPathUtils::copyPath(src_path, new_path);
    }
}
```

create_unit.cpp

```
#include "./include/utils/create_unit.h"

QString CreationUtils::create_unit(QString text, QString path, bool isFolder)
{
    QString absolutePath = path;
    path.append("/");
    path.append(text);
    if (isFolder) {
        return create_folder(text, path, absolutePath);
    } else {
        return create_file(text, path, absolutePath);
    }
}

QString CreationUtils::create_file(QString text, QString path, QString absolutePath)
```

```
{
    int counter = 1;
    QFile newFile(path);

    if (newFile.exists()) {
        while (true) {
            path = prepare_path(absolutePath, text, counter);
            QFile newFile(path);
            if (!newFile.exists()) {
                newFile.open(QFile::WriteOnly);
                break;
            }
            counter++;
        }
    } else {
        newFile.open(QFile::WriteOnly);
    }
    newFile.close();
    return path;
}

QString CreationUtils::create_folder(QString text, QString path, QString absolutePath)
{
    int counter = 1;
    QDir newFolder(path);
    QString pathCopy = path;
    if (newFolder.exists()) {
        while (true) {
            path = prepare_path(absolutePath, text, counter);
            QDir newFolder(path);
            if (!newFolder.exists()) {
                newFolder.mkpath(".");
                break;
            }
            counter++;
        }
    } else {
        newFolder.mkpath(".");
    }
    return path;
}

void CreationUtils::create_folder_from_files(QString path) {
    for (auto &c_file : copiedFiles) {
        QFileInfo copy_info = model_1->fileInfo(c_file);
        if (copy_info.isDir() && !copy_info.isSymLink()) {
            CopyPasteUtils::paste_unit(copy_info, path, true);
        } else {
            CopyPasteUtils::paste_unit(copy_info, path, false);
        }
        if (to_cut) {
            if (model_1->fileInfo(c_file).isDir() && !model_1->fileInfo(c_file).isSymLink()) {
                QDir  dir(model_1->fileInfo(c_file).absoluteFilePath());
                dir.removeRecursively();
            } else {
                QFile file(model_1->fileInfo(c_file).absoluteFilePath());
                file.remove();
```

```
            }
        }
    }
    to_cut = false;
    copiedFiles.clear();
}

QString CreationUtils::prepare_path(QString path, QString text, int counter)
{
    path.append("/");
    QStringList text_list = text.split('.');
    text_list[0].append("(");
    text_list[0].append(QString::number(counter));
    text_list[0].append(")");
    path.append(text_list.join("."));
    return path;
}
```

delete_unit.cpp

```
#include <./include/utils/delete_unit.h>

void DeletionUtils::delete_unit(QString absolutePath, QMessageBox::StandardButton reply, QFileInfo fileInfo)
{
    if (fileInfo.isFile()) {
        delete_file(absolutePath, reply, fileInfo);
    } else if (fileInfo.isSymLink()) {
        delete_file(absolutePath, reply, fileInfo);
    } else {
        delete_folder(absolutePath, reply, fileInfo);
    }
}

void DeletionUtils::delete_file(QString absolutePath, QMessageBox::StandardButton reply, QFileInfo fileInfo)
{
    QFile file(absolutePath);
    if (reply == QMessageBox::Yes) {
        QFileInfo f(file);
        QString own = f.owner();
        if (f.permission(QFile::WriteUser) && own != "root") {
            file.remove();
        } else {
            QMessageBox messageBox;
            messageBox.setText("No permission");
            messageBox.setFixedSize(500, 200);
            messageBox.exec();
        }
    }
}

void DeletionUtils::delete_folder(QString absolutePath, QMessageBox::StandardButton reply, QFileInfo fileInfo)
{
    QDir dir(absolutePath);
    if (reply == QMessageBox::Yes) {
        QFileInfo f(dir.absolutePath());
        QString own = f.owner();
        if (f.permission(QFile::WriteUser) && own != "root") {
```

```
            dir.removeRecursively();
        } else {
            QMessageBox messageBox;
            messageBox.setText("No permission");
            messageBox.setFixedSize(500, 200);
            messageBox.exec();
        }
    }
}
```

dir_size.cpp

```
#include "./include/utils/dir_size.h"

void DirectorySizeCalculationUtils::dirSizeWrap(QString dirPath, Properties *properties, PropertiesWindow *widget)
{
    dirSize(dirPath, properties, widget);
    directorySize = 0;
}

void DirectorySizeCalculationUtils::dirSize(QString dirPath, Properties *properties, PropertiesWindow *widget)
{
    bool wasActive = false;
    QDir dir(dirPath);
    QDir::Filters fileFilters = QDir::Files | QDir::System | QDir::Hidden;
    for (QString filePath : dir.entryList(fileFilters)) {
        QFileInfo fi(dir, filePath);
        directorySize += fi.size();
    }
    QDir::Filters dirFilters = QDir::Dirs | QDir::NoDotAndDotDot | QDir::System | QDir::Hidden;
    for (QString childDirPath : dir.entryList(dirFilters)) {
        properties->setSize(formatSize(directorySize));
        if (widget->isActiveWindow()) {
            wasActive = true;
        }
        if (!widget->isActiveWindow() && wasActive) {
            return;
        }
        emit widget->changeTextSignal(properties->toString());
        dirSize(dirPath + QDir::separator() + childDirPath, properties, widget);
    }
}

QString DirectorySizeCalculationUtils::formatSize(qint64 size)
{
    QStringList units = {"Bytes", "KB", "MB", "GB", "TB", "PB"};
    int i;
    double outputSize = size;
    for (i = 0; i < units.size() - 1; i++) {
        if (outputSize < 1024) break;
        outputSize = outputSize / 1024;
    }
    return QString("%0 %1").arg(outputSize, 0, 'f', 2).arg(units[i]);
}

QString DirectorySizeCalculationUtils::getUnit(qint64 size)
{
```

```
    QStringList units = {"Bytes", "KB", "MB", "GB", "TB", "PB"};
    int i;
    double outputSize = size;
    for (i = 0; i < units.size() - 1; i++) {
        if (outputSize < 1024) break;
        outputSize = outputSize / 1024;
    }
    return units[i];
}

QString DirectorySizeCalculationUtils::getFormattedSize(qint64 size)
{
    QStringList units = {"Bytes", "KB", "MB", "GB", "TB", "PB"};
    int i;
    double outputSize = size;
    for (i = 0; i < units.size() - 1; i++) {
        if (outputSize < 1024) break;
        outputSize = outputSize / 1024;
    }
    return QString("%0").arg(outputSize, 0, 'f', 2);
}
```

filename.cpp

```
#include <./include/utils/filename.h>

QString FileNameUtils::get_filename(QString absolute_path)
{
    return absolute_path.split("/").last();
}
```

navigation.cpp

```
#include <./include/utils/navigation.h>
#include "QDebug"
void NavigationUtils::open_folder(QFileSystemModel *model, QTableView *listView, QLineEdit *lineEdit, QFileInfo
fileInfo)
{
    lineEdit->setText(fileInfo.absoluteFilePath());
    if (fileInfo.fileName() == "..") {
        QDir dir = fileInfo.dir();
        dir.cdUp();
        listView->setRootIndex(model->index(dir.absolutePath()));
    } else if (fileInfo.fileName() == ".") {
        listView->setRootIndex(model->index(""));
    } else if (fileInfo.isDir()) {
        listView->setRootIndex(model->index(fileInfo.absoluteFilePath()));
    }
}
```

rename_unit.cpp

```
#include <./include/utils/rename_unit.h>

void RenameUtils::rename_unit(QFileInfo info, QString text, bool isFolder)
{
    if (isFolder) {
```

```
        rename_folder(info, text);
    } else {
        rename_file(info, text);
    }
}

void RenameUtils::rename_file(QFileInfo info, QString text)
{
    int counter = 1;
    QFile file(info.absoluteFilePath());
    QString new_path = info.absolutePath();
    new_path.append("/");
    new_path.append(text);
    while (!file.rename(new_path)) {
        new_path = info.absolutePath();
        new_path.append("/");
        QStringList text_list = text.split('.');
        text_list[0].append("(");
        text_list[0].append(QString::number(counter));
        text_list[0].append(")");
        new_path.append(text_list.join("."));
        counter++;
    }
}

void RenameUtils::rename_folder(QFileInfo info, QString text)
{
    int counter = 1;
    QDir dir(info.absoluteFilePath());
    QString new_path = info.absolutePath();
    new_path.append("/");
    new_path.append(text);
    while (!dir.rename(info.absoluteFilePath(), new_path)) {
        new_path = info.absolutePath();
        new_path.append("/");
        QStringList text_list = text.split('.');
        text_list[0].append("(");
        text_list[0].append(QString::number(counter));
        text_list[0].append(")");
        new_path.append(text_list.join("."));
        counter++;
    }
}
```

search_filter.cpp

```
#include <./include/utils/search_filter.h>

void FiltersUtils::search_filter(QString item_name, QTableView *listView, QLineEdit *lineEdit, QFileSystemModel *model)
{
    QStringList filters;
    QString request = "*";
    request.append(item_name);
    request.append("*");
    filters << request;
    filters << ".";
    filters << "..";
```

```
    model->setNameFilters(filters);
    model->setNameFilterDisables(false);
    listView->setRootIndex(model->index(lineEdit->text()));
}
```

statistics.cpp

```
#include "./include/utils/statistics.h"

const QStringList StatisticsUtils::audio = {"aif", "cda", "mid", "midi", "mp3", "mpa", "ogg", "wav", "wma", "wpl"};
const QStringList StatisticsUtils::archive = {"7z", "arj", "deb", "pkg", "rar", "rpm", "gz", "z", "zip"};
const QStringList StatisticsUtils::data = {"csv", "dat", "db", "log", "mdb", "sav", "sql", "xml"};
const QStringList StatisticsUtils::executable = {"apk", "bat", "bin", "com", "exe", "gadget", "jar", "msi", "wsf"};
const QStringList StatisticsUtils::image = {"ai", "bmp", "gif", "ico", "jpeg", "jpg", "png", "ps", "psd", "svg", "tif", "tiff"};
const QStringList StatisticsUtils::programming_file = {"c", "cgi", "pl", "class", "cpp", "cs", "h", "java", "php", "py", "sh",
"swift", "vb"};
const QStringList StatisticsUtils::system = {"bak", "cab", "cfg", "cpl", "cur", "dll", "dmp", "drv", "icns", "ico", "ini", "lnk",
"msi", "sys", "tmp"};
const QStringList StatisticsUtils::video = {"3g2", "3gp", "avi", "flv", "h264", "m4v", "mkv", "mov", "mp4", "mpg", "mpeg",
"rm", "swf", "vob", "wmv"};
const QStringList StatisticsUtils::text = {"doc", "docx", "odt", "pdf", "rtf", "tex", "txt", "wpd"};

QString StatisticsUtils::extension_type_map(QFileInfo file)
{
    if ( file.isDir()) {
        return "Folder";
    }
    QString extension = file.suffix();
    if (audio.contains(extension, Qt::CaseInsensitive))
        return "Audio";
    else if (archive.contains(extension, Qt::CaseInsensitive))
        return "Archive";
    else if (data.contains(extension, Qt::CaseInsensitive))
        return "Data";
    else if (executable.contains(extension, Qt::CaseInsensitive))
        return "Executable";
    else if (image.contains(extension, Qt::CaseInsensitive))
        return "Image";
    else if (programming_file.contains(extension, Qt::CaseInsensitive))
        return "Programming file";
    else if (system.contains(extension, Qt::CaseInsensitive))
        return "System";
    else if (video.contains(extension, Qt::CaseInsensitive))
        return "Video";
    else if (text.contains(extension, Qt::CaseInsensitive))
        return "Text";
    else
        return "Other";
}

QMap<QString, int> StatisticsUtils::get_types_distribution_by_count(QFileInfo directory)
{
    QDir dir(directory.absoluteFilePath());
    QMap<QString, int> typesMap;
    QFileInfoList entries = dir.entryInfoList(QDir::NoDotAndDotDot | QDir::AllEntries);
    for (QFileInfo entry : entries) {
        QString type = extension_type_map(entry);
```

```
        if (typesMap.contains(type)) {
            typesMap[type]++;
        } else {
            typesMap[type] = 1;
        }
    }
    return typesMap;
}

QMap<QString, qint64> StatisticsUtils::get_types_distribution_by_size(QFileInfo directory)
{
    QDir dir(directory.absoluteFilePath());
    QMap<QString, qint64> typesMap;
    QFileInfoList entries = dir.entryInfoList(QDir::NoDotAndDotDot | QDir::AllEntries);
    for (QFileInfo entry : entries) {
        QString type = extension_type_map(entry);
        if (!entry.isDir()) {
            if (typesMap.contains(type)) {
                typesMap[type] += entry.size();
            } else {
                typesMap[type] = entry.size();
            }
        }
    }
    return typesMap;
}

QList<QStorageInfo> StatisticsUtils::get_storage_info()
{
    return QStorageInfo::root().mountedVolumes();
}
```

swap_drives.cpp

```
#include "./include/utils/swap_drives.h"

void SwapDrivesUtils::configure_ui(QComboBox *ui)
{
    ui->setVisible(false);
#if defined(_WIN32)
    ui->setVisible(true);
    ui->setFocusPolicy(Qt::NoFocus);
    for (auto drive : get_drives_name()) {
        ui->addItem(drive);
    }
#endif
}

QStringList SwapDrivesUtils::get_drives_name()
{
    QStringList list;
    for (auto info : QDir::drives()) {
        list.append(info.absoluteFilePath());
    }
    return list;
}
```

cloud_drive_entity.cpp

```
#include "./include/cloud_drive_entity.h"

void CloudDrive::setName(QString name)
{
   this->name = name;
}

QString CloudDrive::getName()
{
   return name;
}

void CloudDrive::setPath(QString path)
{
   this->path = path;
}

QString CloudDrive::getPath()
{
   return path;
}

void CloudDrive::setSize(qint64 size)
{
   this->size = size;
}

qint64 CloudDrive::getSize()
{
   return size;
}

void CloudDrive::setImage(QPixmap image)
{
   this->image = image;
}

QPixmap CloudDrive::getImage()
{
   return image;
}
```

clouddrivewidget.cpp

```
#include "./include/clouddrivewidget.h"

CloudDriveWidget::CloudDriveWidget(QWidget *parent, QString name, QPixmap image, QString size) :
   QWidget(parent),
   ui(new Ui::CloudDriveWidget)
{
   ui->setupUi(this);
   defaultMode = true;
   show_hide_details_labels(false);
   ui->driveName->setText(name);
   ui->userName->setText(QDir::home().dirName());
```

```
    ui->driveSize->setText(size);
    ui->driveLogo->setPixmap(image);
    connect(this, SIGNAL(clicked(QString)), parent, SLOT(open_cloud_drive(QString)));
    connect(ui->cloudDriveDetailsButton, SIGNAL(clicked()), this, SLOT(open_close_details()));
}

CloudDriveWidget::~CloudDriveWidget()
{
    delete ui;
}

void CloudDriveWidget::mousePressEvent(QMouseEvent *event)
{
    emit clicked(QDir::homePath() + "/" + ui->driveName->text());
}

void CloudDriveWidget::paintEvent(QPaintEvent *)
{
    QStyleOption opt;
    opt.init(this);
    QPainter p(this);
    style()->drawPrimitive(QStyle::PE_Widget, &opt, &p, this);
}

void CloudDriveWidget::open_close_details()
{
    if (defaultMode) {
        ui->driveName->setVisible(false);
        ui->userName->setVisible(false);
        ui->driveSize->setVisible(false);
        ui->driveLogo->setVisible(false);
        show_hide_details_labels(true);
        QMap<QString, qint64> distribution = StatisticsUtils::get_types_distribution_by_size(QFileInfo(QDir::homePath() + "/"
+ ui->driveName->text()));
        ui->docsInfo->setText("Documents\n" + DirectorySizeCalculationUtils::formatSize(distribution.value("Text")));
        ui->photoInfo->setText("Photo\n" + DirectorySizeCalculationUtils::formatSize(distribution.value("Image")));
        ui->musicInfo->setText("Music\n" + DirectorySizeCalculationUtils::formatSize(distribution.value("Audio")));
        ui->otherInfo->setText("Other\n" + DirectorySizeCalculationUtils::formatSize(distribution.value("Other") +
distribution.value("Video") + distribution.value("System") +
                        distribution.value("Programming file") + distribution.value("Executable") + distribution.value("Data") +
distribution.value("Archive")));
        ui->cloudDriveDetailsButton->setText("Close");
        defaultMode = false;
    } else {
        ui->driveName->setVisible(true);
        ui->userName->setVisible(true);
        ui->driveSize->setVisible(true);
        ui->driveLogo->setVisible(true);
        show_hide_details_labels(false);
        ui->cloudDriveDetailsButton->setText("Details");
        defaultMode = true;
    }
}

void CloudDriveWidget::show_hide_details_labels(bool isVisible) const
{
    ui->docsLogo->setVisible(isVisible);
```

```
    ui->docsInfo->setVisible(isVisible);
    ui->photoLogo->setVisible(isVisible);
    ui->photoInfo->setVisible(isVisible);
    ui->musicLogo->setVisible(isVisible);
    ui->musicInfo->setVisible(isVisible);
    ui->otherLogo->setVisible(isVisible);
    ui->otherInfo->setVisible(isVisible);
}
```

discview.cpp

```
#include "./include/discview.h"

DiscView::DiscView(QWidget *parent, QString discName, QString systemType, QString currentMemory, QString
currentMetrics, QString totalMemory, QString totalMetrics, QString ready, QString readOnly) :
    QWidget(parent),
    ui(new Ui::DiscView)
{
    ui->setupUi(this);
    ui->horizontalLayout->setAlignment(Qt::AlignLeft);
    ui->discName->setText(discName);
    ui->systemType->setText(systemType);
    ui->currentMemory->setText(currentMemory);
    ui->currentMetrics->setText(currentMetrics);
    ui->totalMemory->setText(totalMemory);
    ui->totalMetrics->setText(totalMetrics);
    ui->isReadOnly->setText(readOnly);
    ui->isReady->setText(ready);
}

DiscView::~DiscView()
{
    delete ui;
}
```

donutbreakdownchart.cpp

```
#include "./include/donutbreakdownchart.h"
#include "./include/mainslice.h"
#include <QtCharts/QPieSlice>
#include <QtCharts/QPieLegendMarker>

QT_CHARTS_USE_NAMESPACE

DonutBreakdownChart::DonutBreakdownChart(QGraphicsItem *parent, Qt::WindowFlags wFlags)
    : QChart(QChart::ChartTypeCartesian, parent, wFlags)
{
    m_mainSeries = new QPieSeries();
    m_mainSeries->setPieSize(0.7);
    QChart::addSeries(m_mainSeries);
}

void DonutBreakdownChart::addBreakdownSeries(QPieSeries *breakdownSeries, QColor color)
{
    QFont font("Arial", 8);

    MainSlice *mainSlice = new MainSlice(breakdownSeries);
```

```
    mainSlice->setName(breakdownSeries->name());
    mainSlice->setValue(breakdownSeries->sum());
    m_mainSeries->append(mainSlice);

    mainSlice->setBrush(color);
    mainSlice->setLabelVisible();
    mainSlice->setLabelColor(Qt::white);
    mainSlice->setLabelPosition(QPieSlice::LabelInsideHorizontal);
    mainSlice->setLabelFont(font);

    breakdownSeries->setPieSize(0.8);
    breakdownSeries->setHoleSize(0.7);
    const auto slices = breakdownSeries->slices();
    for (QPieSlice *slice : slices) {
        slice->setLabelFont(font);
    }
    slices[0]->setColor(Qt::gray);
    slices[1]->setColor(QColor("#0066CC"));
    QChart::addSeries(breakdownSeries);
    recalculateAngles();
    updateLegendMarkers();
}

void DonutBreakdownChart::recalculateAngles() const
{
    qreal angle = 0;
    const auto slices = m_mainSeries->slices();
    for (QPieSlice *slice : slices) {
        QPieSeries *breakdownSeries = qobject_cast<MainSlice *>(slice)->breakdownSeries();
        breakdownSeries->setPieStartAngle(angle);
        angle += slice->percentage() * 360.0;
        breakdownSeries->setPieEndAngle(angle);
    }
}

void DonutBreakdownChart::updateLegendMarkers() const
{
    const auto allseries = series();
    for (QAbstractSeries *series : allseries) {
        const auto markers = legend()->markers(series);
        for (QLegendMarker *marker : markers) {
            QPieLegendMarker *pieMarker = qobject_cast<QPieLegendMarker *>(marker);
            if (series == m_mainSeries) {
                pieMarker->setVisible(false);
            } else {
                pieMarker->setLabel(QString("%1 %2%")
                                    .arg(pieMarker->slice()->label())
                                    .arg(pieMarker->slice()->percentage() * 100, 0, 'f', 2));
                pieMarker->setFont(QFont("Arial", 8));
            }
        }
    }
}
```

favoritepathscontainer.cpp

```
#include "./include/favoritepathscontainer.h"
```

```
FavoritePathsContainer::FavoritePathsContainer(QWidget *parent, QString defaultPath) :
    QWidget(parent),
    ui(new Ui::FavoritePathsContainer)
{
    ui->setupUi(this);
    ui->contentLayout->setAlignment(Qt::AlignTop);
    ui->lineEdit->setText(defaultPath);
    connect(ui->lineEdit, SIGNAL(returnPressed()), this, SLOT(add_favorite_path()));
    connect(ui->favoritesPushButton, SIGNAL(clicked()), this, SLOT(add_favorite_path()));
    for (QString path : existingFavoritePaths) {
        FavoritePathWidget *newPath = new FavoritePathWidget(this, path);
        if (newPath->validate(path)) {
            ui->contentLayout->addWidget(newPath);
        }
    }
}

FavoritePathsContainer::~FavoritePathsContainer()
{
    delete ui;
}

void FavoritePathsContainer::add_favorite_path()
{
    QString path = ui->lineEdit->text();
    FavoritePathWidget *newPath = new FavoritePathWidget(this, path);
    if (newPath->validate(path) && !existingFavoritePaths.contains(path)) {
        ui->lineEdit->setText("");
        ui->contentLayout->addWidget(newPath);
        existingFavoritePaths.append(path);
    }
}
```

favoritepathwidget.cpp

```
#include "./include/favoritepathwidget.h"

FavoritePathWidget::FavoritePathWidget(QWidget *parent, QString path) :
    QWidget(parent),
    ui(new Ui::FavoritePathWidget)
{
    ui->setupUi(this);
    ui->okButton->setVisible(false);
    ui->lineEdit->setVisible(false);
    connect(ui->deleteButton, SIGNAL(clicked()), this, SLOT(delete_widget()));
    connect(this, SIGNAL(clicked(QString)), parent->parentWidget()->parentWidget(), SLOT(open_favorite_path(QString)));
    connect(this, SIGNAL(clicked(QString)), parent->parentWidget(), SLOT(close_window()));
    connect(ui->editButton, SIGNAL(clicked()), this, SLOT(edit_path()));
    connect(ui->lineEdit, SIGNAL(returnPressed()), this, SLOT(save_path()));
    connect(ui->okButton, SIGNAL(clicked()), this, SLOT(save_path()));
    ui->favoritePathWidgetLabel->setText(path);
}

FavoritePathWidget::~FavoritePathWidget()
{
    delete ui;
```

```
}

void FavoritePathWidget::delete_widget()
{
   this->deleteLater();
   existingFavoritePaths.removeOne(ui->favoritePathWidgetLabel->text());
}

void FavoritePathWidget::mousePressEvent(QMouseEvent *event)
{
   emit clicked(ui->favoritePathWidgetLabel->text());
}

void FavoritePathWidget::paintEvent(QPaintEvent *)
{
   QStyleOption opt;
   opt.init(this);
   QPainter p(this);
   style()->drawPrimitive(QStyle::PE_Widget, &opt, &p, this);
}

void FavoritePathWidget::edit_path() const
{
   ui->lineEdit->setText(ui->favoritePathWidgetLabel->text());
   ui->lineEdit->selectAll();
   ui->favoritePathWidgetLabel->setVisible(false);
   ui->lineEdit->setVisible(true);
   ui->okButton->setVisible(true);
   ui->lineEdit->setFocus();
}

void FavoritePathWidget::save_path() const
{
   if (validate(ui->lineEdit->text()) && !existingFavoritePaths.contains(ui->lineEdit->text())) {
      existingFavoritePaths.append(ui->lineEdit->text());
      existingFavoritePaths.removeOne(ui->favoritePathWidgetLabel->text()); ui-
      >favoritePathWidgetLabel->setText(ui->lineEdit->text());
      ui->lineEdit->setVisible(false);
      ui->lineEdit->clear();
      ui->okButton->setVisible(false);
      ui->favoritePathWidgetLabel->setVisible(true);
   } else if (ui->lineEdit->text() == ui->favoritePathWidgetLabel->text()) {
      ui->lineEdit->setVisible(false);
      ui->lineEdit->clear();
      ui->okButton->setVisible(false);
      ui->favoritePathWidgetLabel->setVisible(true);
   }
}

bool FavoritePathWidget::validate(QString path) const
{
   if (QDir(path).exists() && !path.isEmpty()) {
      return true;
   }
   return false;
}
```

favoritesmainwindow.cpp

```
#include "./include/favoritesmainwindow.h"

FavoritesMainWindow::FavoritesMainWindow(QWidget *parent) : QMainWindow(parent)
{

}

void FavoritesMainWindow::closeEvent(QCloseEvent *event)
{
   ConfigParser::change_config("favorites", ConfigParser::list_to_string(existingFavoritePaths));
   event->accept();
}

void FavoritesMainWindow::close_window()
{
   close();
}
```

main.cpp

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

int main(int argc, char *argv[])
{
   QApplication a(argc, argv);
   a.setWindowIcon(QIcon(":/icons/icon.png"));
   MainWindow w;
   w.show();
   return a.exec();
}
```

mainslice.cpp

```
#include "./include/mainslice.h"

QT_CHARTS_USE_NAMESPACE

MainSlice::MainSlice(QPieSeries *breakdownSeries, QObject *parent)
   : QPieSlice(parent),
     m_breakdownSeries(breakdownSeries)
{
   connect(this, &MainSlice::percentageChanged, this, &MainSlice::updateLabel);
}

QPieSeries *MainSlice::breakdownSeries() const
{
   return m_breakdownSeries;
}

void MainSlice::setName(QString name)
{
   m_name = name;
}
```

```
QString MainSlice::name() const
{
   return m_name;
}

void MainSlice::updateLabel()
{
   this->setLabel(QString("%1").arg(m_name));
}
```

properties.cpp

```
#include "./include/properties.h"

void Properties::setName(QString name)
{
   this->name = name;
}

QString Properties::getName()
{
   return name;
}

void Properties::setType(QString type)
{
   this->type = type;
}

QString Properties::getType()
{
   return type;
}

void Properties::setSize(QString size)
{
   this->size = size;
}

QString Properties::getSize()
{
   return size;
}

void Properties::setParentFolder(QString parent_folder)
{
   this->parent_folder = parent_folder;
}

QString Properties::getParentFolder()
{
   return parent_folder;
}

void Properties::setGroup(QString group)
{
```

```
    this->group = group;
}

QString Properties::getGroup()
{
    return group;
}

void Properties::setOwner(QString owner)
{
    this->owner = owner;
}

QString Properties::getOwner()
{
    return owner;
}

void Properties::setCreated(QString created)
{
    this->created = created;
}

QString Properties::getCreated()
{
    return created;
}

void Properties::setLastModified(QString last_modified)
{
    this->last_modified = last_modified;
}

QString Properties::getLastModified()
{
    return last_modified;
}

QString Properties::toString()
{
    QString properties_text;
    QStringList properties_list = {"Name: ", "Type: ", "Size: ", "Parent folder: ",
                                   "Group: ", "Owner: ", "Created: ", "Last modified: "
                                  };
    QStringList values_list = {name, type, size, parent_folder,
                               group, owner, created, last_modified
                              };
    for (int i = 0; i < properties_list.size(); i++) {
        properties_text.append(properties_list[i]);
        properties_text.append(values_list[i]);
        properties_text.append("\n\n");
    }
    return properties_text;
}
```

propertieswindow.cpp

```
#include "./include/propertieswindow.h"

PropertiesWindow::PropertiesWindow(QWidget *parent)
   : QTextEdit(parent)
{

}

void PropertiesWindow::changeTextSlot(const QString &message)
{
   this->setText(message);
}
```

searchwindow.cpp

```
#include "./include/searchwindow.h"

enum { absoluteFileNameRole = Qt::UserRole + 1 };

static inline QString fileNameOfItem(const QTableWidgetItem *item)
{
   return item->data(absoluteFileNameRole).toString();
}

static inline void openFile(const QString &fileName)
{
   QDesktopServices::openUrl(QUrl::fromLocalFile(fileName));
}

SearchWindow::~SearchWindow()
{
   QLayoutItem *item;
   while ( ( item = mainLayout->takeAt( 0 ) ) != NULL )
   {
      delete item->widget();
      delete item;
   }
   delete mainLayout;
}

SearchWindow::SearchWindow(QWidget *parent)
   : QWidget(parent)
{
   setWindowTitle(tr("Find Files"));
   findButton = new QPushButton(tr("&Find"), this);
   connect(findButton, &QAbstractButton::clicked, this, &SearchWindow::find);

   fileComboBox = createComboBox(tr("*"));
   connect(fileComboBox->lineEdit(), &QLineEdit::returnPressed,
         this, &SearchWindow::animateFindClick);
   textComboBox = createComboBox();
   connect(textComboBox->lineEdit(), &QLineEdit::returnPressed,
         this, &SearchWindow::animateFindClick);
   directoryComboBox = createComboBox();
   connect(directoryComboBox->lineEdit(), &QLineEdit::returnPressed,
         this, &SearchWindow::animateFindClick);
```

```
    filesFoundLabel = new QLabel;

    createFilesTable();

    mainLayout = new QGridLayout(this);
    mainLayout->addWidget(new QLabel(tr("Named:")), 0, 0);
    mainLayout->addWidget(fileComboBox, 0, 1, 1, 2);
    mainLayout->addWidget(new QLabel(tr("Containing text:")), 1, 0);
    mainLayout->addWidget(textComboBox, 1, 1, 1, 2);
    mainLayout->addWidget(new QLabel(tr("In directory:")), 2, 0);
    mainLayout->addWidget(directoryComboBox, 2, 1, 1, 2);
    mainLayout->addWidget(filesTable, 3, 0, 1, 3);
    mainLayout->addWidget(filesFoundLabel, 4, 0, 1, 2);
    mainLayout->addWidget(findButton, 4, 2);
    connect(new QShortcut(QKeySequence::Quit, this), &QShortcut::activated,
            qApp, &QApplication::quit);
}

static void updateComboBox(QComboBox *comboBox)
{
    if (comboBox->findText(comboBox->currentText()) == -1)
        comboBox->addItem(comboBox->currentText());
}

void SearchWindow::find()
{
    filesTable->setRowCount(0);
    findButton->disconnect();
    std::thread worker([this]() {
        QString fileName = fileComboBox->currentText();
        QString text = textComboBox->currentText();
        QString path = QDir::cleanPath(directoryComboBox->currentText());
        currentDir = QDir(path);

        updateComboBox(fileComboBox);
        updateComboBox(textComboBox);
        updateComboBox(directoryComboBox);

        QStringList filter;
        if (!fileName.isEmpty())
            filter << fileName;
        QDirIterator it(path, filter, QDir::AllEntries | QDir::NoSymLinks | QDir::NoDotAndDotDot,
QDirIterator::Subdirectories);
        findFiles(it, text);
    });
    worker.detach();
}

void SearchWindow::animateFindClick() const
{
    findButton->animateClick();
}

void SearchWindow::findFiles(QDirIterator &iterator, const QString &text) const
{
    QStringList files;
    while (iterator.hasNext()) {
```

```
        QString path = iterator.next();
        if (!QFileInfo(path).isDir()) {
            if (text.isEmpty()) {
                showFile(path);
            } else {
                files << path;
            }
        }
    }
    if (!text.isEmpty()) {
        QMimeDatabase mimeDatabase;
        QStringList foundFiles;

        for (int i = 0; i < files.size(); ++i) {
            QCoreApplication::processEvents();
            const QString fileName = files.at(i);
            const QMimeType mimeType = mimeDatabase.mimeTypeForFile(fileName);
            if (mimeType.isValid() && !mimeType.inherits(QStringLiteral("text/plain"))) {
                continue;
            }
            QFile file(fileName);
            if (file.open(QIODevice::ReadOnly)) {
                QString line;
                QTextStream in(&file);
                while (!in.atEnd()) {
                    line = in.readLine();
                    if (line.contains(text, Qt::CaseInsensitive)) {
                        foundFiles << files[i];
                        showFile(fileName);
                        break;
                    }
                }
            }
        }
    }
    connect(findButton, &QAbstractButton::clicked, this, &SearchWindow::find);
}

void SearchWindow::showFile(const QString &filePath) const
{
    const QString toolTip = QDir::toNativeSeparators(filePath);
    const QString relativePath = QDir::toNativeSeparators(currentDir.relativeFilePath(filePath));
    const qint64 size = QFileInfo(filePath).size();
    QTableWidgetItem *fileNameItem = new QTableWidgetItem(relativePath);
    fileNameItem->setData(absoluteFileNameRole, QVariant(filePath));
    fileNameItem->setToolTip(toolTip);
    fileNameItem->setFlags(fileNameItem->flags() ^ Qt::ItemIsEditable);
    QTableWidgetItem *sizeItem = new QTableWidgetItem(QLocale().formattedDataSize(size));
    sizeItem->setData(absoluteFileNameRole, QVariant(filePath));
    sizeItem->setToolTip(toolTip);
    sizeItem->setTextAlignment(Qt::AlignRight | Qt::AlignVCenter);
    sizeItem->setFlags(sizeItem->flags() ^ Qt::ItemIsEditable);

    int row = filesTable->rowCount();
    filesTable->insertRow(row);
    filesTable->setItem(row, 0, fileNameItem);
    filesTable->setItem(row, 1, sizeItem);
```

```
    filesFoundLabel->setText(tr("%n file(s) found (Double click on a file to open it)", nullptr, filesTable->rowCount()));
    filesFoundLabel->setWordWrap(true);
}

QComboBox *SearchWindow::createComboBox(const QString &text) const
{
    QComboBox *comboBox = new QComboBox;
    comboBox->setEditable(true);
    comboBox->addItem(text);
    comboBox->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Preferred);
    return comboBox;
}

void SearchWindow::createFilesTable()
{
    filesTable = new QTableWidget(0, 2);
    filesTable->setSelectionBehavior(QAbstractItemView::SelectRows);

    QStringList labels;
    labels << tr("Filename") << tr("Size");
    filesTable->setHorizontalHeaderLabels(labels);
    filesTable->horizontalHeader()->setSectionResizeMode(0, QHeaderView::Stretch);
    filesTable->verticalHeader()->hide();
    filesTable->setFocusPolicy(Qt::NoFocus);
    filesTable->setShowGrid(false);

    connect(filesTable, &QTableWidget::cellActivated,
            this, &SearchWindow::openFileOfItem);
}

void SearchWindow::openFileOfItem(int row, int /* column */) const
{
    const QTableWidgetItem *item = filesTable->item(row, 0);
    openFile(fileNameOfItem(item));
}
```

statisticswidget.cpp

```
#include "./include/statisticswidget.h"

//QList<QColor> colorTheme = {QColor("#008232"), QColor("#F71100"), QColor("#9C00EB"), QColor("#3AD65C"),
QColor("#000FF6")};
QList<QColor> colorTheme = {QColor("#3e3e3e"), QColor("#3e3e3e"), QColor("#3e3e3e"), QColor("#3e3e3e"),
QColor("#3e3e3e")};

StatisticsWidget::StatisticsWidget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::StatisticsWidget)
{
    ui->setupUi(this);
    ConfigParser::configure();
    currentChartsTheme = currentTheme;
    connect(this, SIGNAL(update_charts(QFileInfo)), this, SLOT(paint_statistics(QFileInfo)));
    QList<QStorageInfo> infoList = StatisticsUtils::get_storage_info();
    donutBreakdown = new DonutBreakdownChart();
    donutBreakdown->setAnimationOptions(QChart::AllAnimations);
    donutBreakdown->legend()->setAlignment(Qt::AlignRight);
```

```
    int colorCounter = 0;
    for (QStorageInfo info : infoList) {
        QPieSeries *series = new QPieSeries();
        series->setName(QString("%1
%2").arg(info.rootPath()).arg(DirectorySizeCalculationUtils::formatSize(info.bytesTotal())));
        series->append("Free space", info.bytesAvailable());
        series->append("Used space", info.bytesTotal() - info.bytesAvailable());
        QPieSlice *used = series->slices().at(0);
        QPieSlice *free = series->slices().at(1);
        used->setLabelVisible(false);
        free->setLabelVisible(false);
        used->setBorderWidth(0);
        free->setBorderWidth(0);
        donutBreakdown->addBreakdownSeries(series, colorTheme[colorCounter]);
        donutBreakdown->legend()->hide();
        colorCounter++;
        if (colorCounter == colorTheme.size()) {
            colorCounter = 0;
        }
    }
    donutBreakdown->setContentsMargins(-60, -60, -60, -60);
    if (currentTheme == Theme::DARK) {
        donutBreakdown->setPlotAreaBackgroundBrush(QBrush(QColor("#2B2C2D")));
    } else if (currentTheme == Theme::LIGHT) {
        donutBreakdown->setPlotAreaBackgroundBrush(QBrush(QColor("#FFFFFF")));
    }
    donutBreakdown->setPlotAreaBackgroundVisible(true);
    ui->totalConsumption->setChart(donutBreakdown);
    ui->totalConsumption->setRenderHint(QPainter::Antialiasing);
    for (int i = 0; i < infoList.size(); i++) {
        QStorageInfo info = infoList[i];
        QString ready = info.isReady() ? "yes" : "no";
        QString readOnly = info.isReadOnly() ? "yes" : "no";
        DiscView *disc = new DiscView(this, "Disc: " + infoList[i].rootPath().remove(info.rootPath().size() - 2,
info.rootPath().size()),
                                      "File system type: " + infoList[i].fileSystemType(),
                                      DirectorySizeCalculationUtils::getFormattedSize(info.bytesTotal() - info.bytesAvailable()) + " ",
                                      DirectorySizeCalculationUtils::getUnit(info.bytesTotal() - info.bytesAvailable()),
                                      DirectorySizeCalculationUtils::getFormattedSize(info.bytesTotal()) + " ",
                                      DirectorySizeCalculationUtils::getUnit(info.bytesTotal()),
                                      "Ready: " + ready,
                                      "ReadOnly: " + readOnly);
        ui->discs->addTab(disc, infoList[i].rootPath().remove(info.rootPath().size() - 2, info.rootPath().size()));
    }
    setup_cloud_drives();
}

StatisticsWidget::~StatisticsWidget()
{
    delete ui;
}

void StatisticsWidget::paint_statistics(QFileInfo info)
{
    QPieSeries *series = new QPieSeries();
    QMapIterator<QString, int> it(StatisticsUtils::get_types_distribution_by_count(info));
    while (it.hasNext()) {
```

```
        it.next();
        series->append(it.key(), it.value());
        series->setLabelsVisible(true);
    }
    typesChart = new QChart();
    typesChart->addSeries(series);
    typesChart->legend()->hide();
    typesChart->setAnimationOptions(QChart::AllAnimations);
    if (currentTheme == Theme::DARK) {
        typesChart->setTheme(QChart::ChartThemeHighContrast);
        for (QPieSlice *slice : series->slices()) {
            slice->setLabelColor(QColor("#BABABF"));
        }
        typesChart->setPlotAreaBackgroundBrush(QBrush(QColor("#2B2C2D")));
    } else if (currentTheme == Theme::LIGHT) {
        typesChart->setTheme(QChart::ChartThemeHighContrast);
        for (QPieSlice *slice : series->slices()) {
            slice->setLabelColor(QColor("#101012"));
        }
        typesChart->setPlotAreaBackgroundBrush(QBrush(QColor("#FFFFFF")));
    }
    typesChart->setContentsMargins(-40, -40, -40, -40);
    typesChart->setPlotAreaBackgroundVisible(true);
    ui->typesDistribution->setChart(typesChart);
    ui->typesDistribution->setRenderHint(QPainter::Antialiasing);
}

void StatisticsWidget::setup_cloud_drives() const
{
    QList<CloudDrive> drives = CloudDriveUtils::get_supported_drives();
    for (auto drive : drives) {
        CloudDriveWidget *driveWidget = new CloudDriveWidget(this->parentWidget()->parentWidget()->parentWidget()-
>parentWidget(), drive.getName(), drive.getImage(), DirectorySizeCalculationUtils::formatSize(drive.getSize()));
        ui->verticalLayout->insertWidget(0, driveWidget);
    }
}

void StatisticsWidget::updateChartsTheme(Theme theme)
{
    if (currentChartsTheme != theme) {
        if (theme == Theme::DARK) {
            typesChart->setTheme(QChart::ChartThemeHighContrast);
            for (QAbstractSeries *series : typesChart->series())
                for (QPieSlice *slice : ((QPieSeries *)series)->slices())
                    slice->setLabelColor(QColor("#BABABF"));
            typesChart->setPlotAreaBackgroundBrush(QBrush(QColor("#2B2C2D")));
            donutBreakdown->setPlotAreaBackgroundBrush(QBrush(QColor("#2B2C2D")));
        } else if (theme == Theme::LIGHT) {
            typesChart->setTheme(QChart::ChartThemeHighContrast);
            for (QAbstractSeries *series : typesChart->series())
                for (QPieSlice *slice : ((QPieSeries *)series)->slices())
                    slice->setLabelColor(QColor("#101012"));
            typesChart->setPlotAreaBackgroundBrush(QBrush(QColor("#FFFFFF")));
            donutBreakdown->setPlotAreaBackgroundBrush(QBrush(QColor("#FFFFFF")));
        }
        currentChartsTheme = theme;
    }
```

```
}
```

mainwindow.cpp

```
#include "./include/mainwindow.h"

#if defined(_WIN32)
QString mPath = "C:/";
#endif
#if defined(unix) || defined(_unix_) || defined(_unix)
QString mPath = "/";
#endif
#if defined(_APPLE_)
QString mPath = "/";
#endif

QModelIndex chosenFile;
QModelIndex copiedFile;
QList<QModelIndex> chosenFiles;
QList<QModelIndex> copiedFiles;
bool to_cut = false;
QFileSystemModel *model_1;
QFileSystemModel *model_2;
qint64 directorySize;
Panel active_panel;
Theme currentTheme;
QStringList existingFavoritePaths;
Panel favorites_active_panel;


MainWindow::MainWindow(QWidget *parent)
   : QMainWindow(parent)
   , ui(new Ui::MainWindow)
{
   ui->setupUi(this);
   existingFavoritePaths = QStringList();
   model_1 = new QFileSystemModel(this);
   model_2 = new QFileSystemModel(this);
   model_1->setFilter(QDir::QDir::AllEntries | QDir::QDir::NoDot);
   model_1->setRootPath(mPath);
   model_2->setFilter(QDir::QDir::AllEntries | QDir::QDir::NoDot);
   model_2->setRootPath(mPath);
   ui->lineEdit_1->setText(mPath);
   ui->lineEdit_2->setText(mPath);
   list_view_init(ui->listView_1, model_1);
   list_view_init(ui->listView_2, model_2);
   ui->search_1->setVisible(false);
   ui->search_2->setVisible(false);
   SwapDrivesUtils::configure_ui(ui->comboBox_1);
   SwapDrivesUtils::configure_ui(ui->comboBox_2);
   configure();
   emit ui->statistics->update_charts(QFileInfo(mPath));
   connect(ui->comboBox_1, SIGNAL(textActivated(QString)), this, SLOT(change_root_path(QString)));
   connect(ui->comboBox_2, SIGNAL(textActivated(QString)), this, SLOT(change_root_path(QString)));
   connect(ui->lineEdit_1, SIGNAL(returnPressed()), SLOT(line_edit_enter()));
   connect(ui->lineEdit_2, SIGNAL(returnPressed()), SLOT(line_edit_enter()));
   connect(ui->search_button_1, SIGNAL (released()), this, SLOT (show_hide_search_1()));
```

```
    connect(ui->search_button_2, SIGNAL (released()), this, SLOT (show_hide_search_2()));
    connect(ui->search_1, SIGNAL(returnPressed()), SLOT(searchEnter()));
    connect(ui->search_2, SIGNAL(returnPressed()), SLOT(searchEnter()));
    connect(ui->actionDark, SIGNAL(triggered()), this, SLOT(change_theme()));
    connect(ui->actionLight, SIGNAL(triggered()), this, SLOT(change_theme()));
    connect(ui->favoritePathsButton_1, SIGNAL(clicked()), this, SLOT(show_favorite_paths()));
    connect(ui->favoritePathsButton_2, SIGNAL(clicked()), this, SLOT(show_favorite_paths()));
    connect(ui->actionSearch, SIGNAL(triggered()), this, SLOT(show_search_window()));
    new QShortcut(QKeySequence(Qt::CTRL + Qt::Key_C), this, SLOT(copy_file()));
    new QShortcut(QKeySequence(Qt::Key_Escape), this, SLOT(close_search()));
//    new QShortcut(QKeySequence(Qt::Key_Delete), this, SLOT(delete_file())); TODO
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::click(const QModelIndex &index)
{
    chosenFile = index.siblingAtColumn(0);
}

void MainWindow::on_listView_doubleClicked(const QModelIndex &index)
{
    this->setCursor(QCursor(Qt::WaitCursor));
    QTableView *listView = (QTableView *)sender();
    QFileInfo fileInfo = model_1->fileInfo(index);
    if (fileInfo.isFile()) {
        QDesktopServices::openUrl(QUrl(fileInfo.absoluteFilePath().prepend("file:///")));
        this->setCursor(QCursor(Qt::ArrowCursor));
        return;
    } else if (fileInfo.isSymLink()) {
        if (listView == ui->listView_1) {
            QModelIndex index = model_1->index(fileInfo.symLinkTarget());
            emit ui->statistics->update_charts(model_1->fileInfo(index));
            NavigationUtils::open_folder(model_1, ui->listView_1, ui->lineEdit_1, model_1->fileInfo(index));
        } else {
            emit ui->statistics->update_charts(model_2->fileInfo(index));
            QModelIndex index = model_2->index(fileInfo.symLinkTarget());
            NavigationUtils::open_folder(model_2, ui->listView_2, ui->lineEdit_2, model_2->fileInfo(index));
        }
        this->setCursor(QCursor(Qt::ArrowCursor));
        return;
    }
    if (listView == ui->listView_1) {
        NavigationUtils::open_folder(model_1, ui->listView_1, ui->lineEdit_1, fileInfo);
    } else {
        NavigationUtils::open_folder(model_2, ui->listView_2, ui->lineEdit_2, fileInfo);
    }
    emit ui->statistics->update_charts(fileInfo);
    this->setCursor(QCursor(Qt::ArrowCursor));
}

void MainWindow::show_hide_search_1()
{
    if (ui->search_1->isVisible()) {
```

```
        ui->search_1->setVisible(false);
    } else {
        ui->search_1->setVisible(true);
    }
}

void MainWindow::show_hide_search_2()
{
    if (ui->search_2->isVisible()) {
        ui->search_2->setVisible(false);
    } else {
        ui->search_2->setVisible(true);
    }
}

void MainWindow::delete_file()
{
    QMessageBox::StandardButton reply;
    QString question = "Are you sure you want to delete " + QString::number(chosenFiles.size()) +
                       " files?";
    reply = QMessageBox::question(this, "Delete", question,
                                  QMessageBox::Yes | QMessageBox::No);
    this->setCursor(QCursor(Qt::WaitCursor));
    for (auto &c_file : chosenFiles) {
        QString absolutePath = model_1->fileInfo(c_file).absoluteFilePath();
        DeletionUtils::delete_unit(absolutePath, reply, model_1->fileInfo(c_file));
    }
    this->setCursor(QCursor(Qt::ArrowCursor));
    chosenFiles.clear();
}

void MainWindow::rename_file()
{
    bool result;
    QFileInfo info = model_1->fileInfo(chosenFile);
    QString name = info.baseName();
    if (!(info.completeSuffix() == "")) {
        name.append(".");
    }
    name.append(info.completeSuffix());
    QString text;
    if (info.permission(QFile::WriteUser)) {
        text = QInputDialog::getText(this, tr("Rename"),
                                     tr("New name:"), QLineEdit::Normal,
                                     name, &result);
    } else {
        QMessageBox msg;
        msg.setText("No permission");
        msg.setFixedSize(500, 200);
        msg.exec();
    }

    if (result) {
        if (info.isDir()) {
            RenameUtils::rename_unit(info, text, true);
        } else {
            RenameUtils::rename_unit(info, text, false);
```

```
        }
    }
}

void MainWindow::get_properties()
{
    this->setCursor(QCursor(Qt::WaitCursor));
    QFileInfo info(ui->lineEdit_1->text());
    if (chosenFile.isValid()) {
        info = model_1->fileInfo(chosenFile);
    } else {
        if (active_panel == Panel::PANEL_1) {
            info = QFileInfo(ui->lineEdit_1->text());
        } else if (active_panel == Panel::PANEL_2) {
            info = QFileInfo(ui->lineEdit_2->text());
        }
    }
    QDialog *widget = new QDialog(this);
    QVBoxLayout *layout = new QVBoxLayout(widget);
    PropertiesWindow *properties_window = new PropertiesWindow(widget);
    properties_window->setReadOnly(true);
    QFont sansFont("Times", 10);
    properties_window->setCurrentFont(sansFont);
    widget->setWindowTitle("Properties");
    QPushButton *button = new QPushButton(widget);
    button->setText("Close");
    connect(button, SIGNAL(clicked()), widget, SLOT(close()));
    Properties *properties = new Properties();
    QString name = info.baseName();
    if (!(info.completeSuffix() == "")) {
        name.append(".");
    }
    name.append(info.completeSuffix());
    properties->setName(name);
    QString type;
    if (info.isDir()) {
        type = "directory";
    } else if (info.isExecutable()) {
        type = "executable";
    } else if (info.isSymLink()) {
        type = "symbolic link";
    } else if (info.isBundle()) {
        type = "bundle";
    } else {
        type = "file";
    }
    properties->setType(type);
    qint64 size;
    if (info.isDir()) {
        directorySize = 0;
        connect(properties_window, SIGNAL(changeTextSignal(QString)), properties_window,
SLOT(changeTextSlot(QString)));
        std::thread worker(DirectorySizeCalculationUtils::dirSizeWrap, info.absoluteFilePath(), properties, properties_window);
        worker.detach();
        size = directorySize;
    } else {
        size = info.size();
```

```
    }
    properties->setSize(DirectorySizeCalculationUtils::formatSize(size));
    properties->setParentFolder(info.absolutePath());
    properties->setGroup(info.group());
    properties->setOwner(info.owner());
    properties->setLastModified(info.lastModified().toString(Qt::SystemLocaleLongDate));
    properties->setCreated(info.created().toString(Qt::SystemLocaleLongDate));
    properties_window->setText(properties->toString());
    properties_window->setFixedHeight(this->height() / 2);
    properties_window->setFixedWidth(this->width() / 4);
    properties_window->setAttribute(Qt::WA_DeleteOnClose);
    this->setCursor(QCursor(Qt::ArrowCursor));
    layout->addWidget(properties_window, 0, 0);
    layout->addWidget(button);
    widget->setLayout(layout);
    widget->exec();
}

void MainWindow::copy_file()
{
    if (chosenFiles.empty()) {
        copiedFiles.append(chosenFile);
    } else {
        copiedFiles = chosenFiles;
    }
}

void MainWindow::cut_file()
{
    if (chosenFiles.empty()) {
        copiedFiles.append(chosenFile);
    } else {
        copiedFiles = chosenFiles;
    }
    to_cut = true;
}

void MainWindow::paste_file()
{
    this->setCursor(QCursor(Qt::WaitCursor));
    for (auto &c_file : copiedFiles) {
        QFileInfo copy_info = model_1->fileInfo(c_file);
        QString path;
        if (active_panel == Panel::PANEL_1) {
            path = ui->lineEdit_1->text();
        } else if (active_panel == Panel::PANEL_2) {
            path = ui->lineEdit_2->text();
        }
        if (copy_info.isDir() && !copy_info.isSymLink()) {
            CopyPasteUtils::paste_unit(copy_info, path, true);
        } else {
            CopyPasteUtils::paste_unit(copy_info, path, false);
        }
        if (to_cut) {
            if (model_1->fileInfo(c_file).isDir() && !model_1->fileInfo(c_file).isSymLink()) {
                QDir  dir(model_1->fileInfo(c_file).absoluteFilePath());
                dir.removeRecursively();
```

```
            } else {
                QFile file(model_1->fileInfo(c_file).absoluteFilePath());
                file.remove();
            }
        }
    }
    to_cut = false;
    copiedFiles.clear();
    this->setCursor(QCursor(Qt::ArrowCursor));
}

void MainWindow::create_file()
{
    bool result;
    QString text = QInputDialog::getText(this, tr("Create"),
                                         tr("New file:"), QLineEdit::Normal,
                                         "", &result);
    if (result) {
        if (active_panel == Panel::PANEL_1) {
            CreationUtils::create_unit(text, ui->lineEdit_1->text(), false);
        } else if (active_panel == Panel::PANEL_2) {
            CreationUtils::create_unit(text, ui->lineEdit_2->text(), false);
        }
    }
}

void MainWindow::create_folder()
{
    bool result;
    QString text = QInputDialog::getText(this, tr("Create"),
                                         tr("New folder:"), QLineEdit::Normal,
                                         "", &result);
    if (result) {
        if (active_panel == Panel::PANEL_1) {
            QString created = CreationUtils::create_unit(text, ui->lineEdit_1->text(), true);
            if (chosenFiles.size() > 1) {
                cut_file();
                CreationUtils::create_folder_from_files(created);
            }
        } else if (active_panel == Panel::PANEL_2) {
            QString created = CreationUtils::create_unit(text, ui->lineEdit_2->text(), true);
            if (chosenFiles.size() > 1) {
                cut_file();
                CreationUtils::create_folder_from_files(created);
            }
        }
    }
}

void MainWindow::custom_menu_requested(const QPoint &pos)
{
    QTableView *listView = (QTableView *)sender();
    QModelIndex index = listView->indexAt(pos);
    if (listView == ui->listView_1) {
        active_panel = Panel::PANEL_1;
    } else if ( listView == ui->listView_2) {
        active_panel = Panel::PANEL_2;
```

```
    }
    QModelIndexList list = listView->selectionModel()->selectedIndexes();
    chosenFiles.clear();
    foreach (const QModelIndex &indexx, list) {
        if (indexx.column() == 0)
            chosenFiles.append(indexx);
    }
    if (!(model_1->fileInfo(index).completeBaseName() == ".")) {
        chosenFile = index;
        QMenu *menu = new QMenu(this);
        menu->setObjectName("menu");
        QFileInfo fileInfo = model_1->fileInfo(index);
        if (chosenFiles.size() < 2 && fileInfo.isFile() && index.isValid()) {
            QAction *open_action = new QAction("Open", this);
            menu->addAction(open_action);
            connect(open_action, SIGNAL(triggered()), this, SLOT(open_file()));
        }
        QMenu *create_menu = menu->addMenu("Create");
        QAction *create_folder_action = new QAction("Folder", this);
        QAction *create_file_action = new QAction("File", this);
        create_menu->addAction(create_folder_action);
        create_menu->addAction(create_file_action);
        if (index.isValid()) {
            QAction *copy_action = new QAction("Copy", this);
            menu->addAction(copy_action);
            QAction *cut_action = new QAction("Cut", this);
            menu->addAction(cut_action);
            connect(copy_action, SIGNAL(triggered()), this, SLOT(copy_file()));
            connect(cut_action, SIGNAL(triggered()), this, SLOT(cut_file()));
        }
        QAction *paste_action = new QAction("Paste", this);
        menu->addAction(paste_action);
        paste_action->setObjectName("paste_action");
        if (index.isValid()) {
            QAction *delete_action = new QAction("Delete", this);
            menu->addAction(delete_action);
            if (chosenFiles.size() < 2) {
                QAction *rename_action = new QAction("Rename", this);
                menu->addAction(rename_action);
                connect(rename_action, SIGNAL(triggered()), this, SLOT(rename_file()));
            }
            QAction *shortcut_action = new QAction("Create shortcut", this);
            menu->addAction(shortcut_action);
            connect(shortcut_action, SIGNAL(triggered()), this, SLOT(create_shortcut()));
            connect(delete_action, SIGNAL(triggered()), this, SLOT(delete_file()));
        }
        if (chosenFiles.size() < 2) {
            QAction *properties_action = new QAction("Properties", this);
            menu->addAction(properties_action);
            connect(properties_action, SIGNAL(triggered()), this, SLOT(get_properties()));
        }
        connect(create_file_action, SIGNAL(triggered()), this, SLOT(create_file()));
        connect(create_folder_action, SIGNAL(triggered()), this, SLOT(create_folder()));
        connect(paste_action, SIGNAL(triggered()), this, SLOT(paste_file()));
        menu->popup(listView->viewport()->mapToGlobal(pos));
    }
}
```

```
void MainWindow::line_edit_enter()
{
   QLineEdit *line = (QLineEdit *)sender();
   QString path = line->text();
   QDir dir(path);
   QModelIndex idx;
   if (line == ui->search_1) {
      idx = model_1->index(path);
   } else {
      idx = model_2->index(path);
   }
   if (dir.exists()) {
      emit ui->statistics->update_charts(QFileInfo(path));
      if (line == ui->lineEdit_1) {
         ui->listView_1->setRootIndex(idx);
      } else {
         ui->listView_2->setRootIndex(idx);
      }
   }
}

void MainWindow::change_root_path(QString path)
{
   QComboBox *box = (QComboBox *)sender();
   emit ui->statistics->update_charts(QFileInfo(path));
   if (box == ui->comboBox_1) {
      ui->listView_1->setRootIndex(model_1->index(path));
      ui->lineEdit_1->setText(path);
   } else if (box == ui->comboBox_2) {
      ui->listView_2->setRootIndex(model_2->index(path));
      ui->lineEdit_2->setText(path);
   }
}

void MainWindow::searchEnter()
{
   QLineEdit *line = (QLineEdit *)sender();
   QString item_name = line->text();
   if (line == ui->search_1) {
      FiltersUtils::search_filter(item_name, ui->listView_1, ui->lineEdit_1, model_1);
   } else if (line == ui->search_2) {
      FiltersUtils::search_filter(item_name, ui->listView_2, ui->lineEdit_2, model_2);
   }
}

void MainWindow::close_search()
{
   ui->search_1->setVisible(false);
   ui->search_2->setVisible(false);
   QStringList filters;
   model_1->setNameFilters(filters);
   model_2->setNameFilters(filters);
}

void MainWindow::show_favorite_paths()
{
```

```
    FavoritesMainWindow *window = new FavoritesMainWindow(this);
    QPushButton *button = (QPushButton *)sender();
    FavoritePathsContainer *container;
    if (button == ui->favoritePathsButton_1) {
        favorites_active_panel = Panel::PANEL_1;
        container = new FavoritePathsContainer(window, ui->lineEdit_1->text());
    } else {
        favorites_active_panel = Panel::PANEL_2;
        container = new FavoritePathsContainer(window, ui->lineEdit_2->text());
    }
    window->setCentralWidget(container);
    window->setFixedHeight(this->height() / 2);
    window->setFixedWidth(this->width() / 4);
    window->setWindowTitle("Favorites");
    window->setAttribute(Qt::WA_DeleteOnClose);
    window->show();
}

void MainWindow::open_file()
{
    this->setCursor(QCursor(Qt::WaitCursor));
    QFileInfo fileInfo = model_1->fileInfo(chosenFile);
    QDesktopServices::openUrl(QUrl(fileInfo.absoluteFilePath().prepend("file:///")));
    this->setCursor(QCursor(Qt::ArrowCursor));
}

void MainWindow::open_cloud_drive(QString path)
{
    emit ui->statistics->update_charts(QFileInfo(path));
    if (active_panel == Panel::PANEL_1) {
        NavigationUtils::open_folder(model_1, ui->listView_1, ui->lineEdit_1, QFileInfo(path));
    } else if (active_panel == Panel::PANEL_2) {
        NavigationUtils::open_folder(model_2, ui->listView_2, ui->lineEdit_2, QFileInfo(path));
    }
}

void MainWindow::create_shortcut()
{
    this->setCursor(QCursor(Qt::WaitCursor));
    for (auto &c_file : chosenFiles) {
        QString absolutePath = model_1->fileInfo(c_file).absoluteFilePath();
        QFile::link(absolutePath, absolutePath + " - Shortcut.lnk");
    }
    this->setCursor(QCursor(Qt::ArrowCursor));
    chosenFiles.clear();
}

void MainWindow::open_favorite_path(QString path)
{
    emit ui->statistics->update_charts(QFileInfo(path));
    if (favorites_active_panel == Panel::PANEL_1) {
        NavigationUtils::open_folder(model_1, ui->listView_1, ui->lineEdit_1, QFileInfo(path));
    } else if (favorites_active_panel == Panel::PANEL_2) {
        NavigationUtils::open_folder(model_2, ui->listView_2, ui->lineEdit_2, QFileInfo(path));
    }
}
```

```
void MainWindow::change_theme()
{
    QAction *action = (QAction *)sender();
    if (action == ui->actionDark) {
        ConfigParser::change_config(QString("theme"), QString("DARK"));
        currentTheme = Theme::DARK;
        ui->statistics->updateChartsTheme(Theme::DARK);
        QFile fileDark(":/ui/styles/dark.qss");
        fileDark.open(QFile::ReadOnly);
        setStyleSheet(fileDark.readAll());
    } else if (action == ui->actionLight) {
        ConfigParser::change_config(QString("theme"), QString("LIGHT"));
        currentTheme = Theme::LIGHT;
        ui->statistics->updateChartsTheme(Theme::LIGHT);
        QFile fileLight(":/ui/styles/light.qss");
        fileLight.open(QFile::ReadOnly);
        setStyleSheet(fileLight.readAll());
    }
}

void MainWindow::show_search_window() {
    QMainWindow *window = new QMainWindow(this);
    SearchWindow *search = new SearchWindow();
    window->setWindowTitle("Search");
    window->setCentralWidget(search);
    window->setFixedHeight(this->height() / 2);
    window->setFixedWidth(this->width() / 4);
    window->setAttribute(Qt::WA_DeleteOnClose);
    window->show();
}

void MainWindow::configure()
{
    setWindowTitle("QExplorer");
    QList<QScreen *> rec = QGuiApplication::screens();
    resize(rec.first()->availableGeometry().width() / 1.5, rec.first()->availableGeometry().height() / 1.5);
    ConfigParser::configure();
    if (currentTheme == Theme::DARK) {
        QFile fileDark(":/ui/styles/dark.qss");
        fileDark.open(QFile::ReadOnly);
        setStyleSheet(fileDark.readAll());
    } else if (currentTheme == Theme::LIGHT) {
        QFile fileLight(":/ui/styles/light.qss");
        fileLight.open(QFile::ReadOnly);
        setStyleSheet(fileLight.readAll());
    }
}

void MainWindow::list_view_init(QTableView *tableView, QFileSystemModel *model)
{
    tableView->setModel(model);
    QModelIndex idx = model->index(model->rootPath());
    tableView->setRootIndex(idx);
    tableView->verticalHeader()->setVisible(false);
    tableView->setContextMenuPolicy(Qt::CustomContextMenu);
    tableView->setSelectionMode(QAbstractItemView::ExtendedSelection);
    tableView->setDragDropMode(QAbstractItemView::DragDrop);
```

```
    tableView->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);
    tableView->horizontalHeader()->setStretchLastSection(true);
    tableView->setSelectionBehavior(QAbstractItemView::SelectRows);
    tableView->setSortingEnabled(true);
    tableView->horizontalHeader()->setSortIndicator(0, Qt::AscendingOrder);
    tableView->setShowGrid(false);
    tableView->setFocusPolicy(Qt::NoFocus);
    tableView->horizontalHeader()->setSectionResizeMode(0, QHeaderView::Stretch);
    tableView->verticalHeader()->hide();
    connect(tableView, SIGNAL(doubleClicked(QModelIndex)), this, SLOT(on_listView_doubleClicked(QModelIndex)));
    connect(tableView, SIGNAL(clicked(QModelIndex)), this, SLOT(click(QModelIndex)));
    connect(tableView, SIGNAL(customContextMenuRequested(QPoint)), this, SLOT(custom_menu_requested(QPoint)));
}
```