

UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

---

# Development of cross-platform file system manager

---

*Author:*  
Vitalii VOLIANSKYI

*Supervisor:*  
Igor BEREZHNYJ

*A thesis submitted in fulfillment of the requirements  
for the degree of Bachelor of Science*

*in the*

Department of Computer Sciences  
Faculty of Applied Sciences



APPLIED  
SCIENCES  
FACULTY ●

Lviv 2021

## Declaration of Authorship

I, Vitalii VOLIANSKYI, declare that this thesis titled, “Development of cross-platform file system manager” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

---

Date:

---

*“Software and cathedrals are much the same – first we build them, then we pray.”*

Sam Redwine

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

**Development of cross-platform file system manager**

by Vitalii VOLIANSKYI

## *Abstract*

Information storage is one of the most important functions of any OS. That is why there exist such concepts as file and file system. In short, files are information objects that contain data or programs, and the file system, respectively, is a way of organizing these objects.

However, since operating system users are often human, it is not enough to store information, it is also necessary to provide a convenient way to work with it. This is where file managers come into play. We use them almost every time we use a computer and therefore they are extremely required to be fast and convenient. This is what this work is about.

## *Acknowledgements*

First of all, I want to sincerely thank Igor Berezhnyj, who made large impact into the project, introduced a lot of new ideas and always helped me with my questions. Thanks to Oleh Farenjuk for the idea of this project and most importantly for the knowledge to put it into life.

Finally, I want to thank my parents and friends, who everyday support me.

# Contents

<b>Declaration of Authorship</b>	<b>i</b>
<b>Abstract</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>iv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation	1
1.2 Task definition	1
<b>2 Existing solutions</b>	<b>2</b>
2.1 Total Commander	2
2.2 Directory Opus	3
2.3 Windows Explorer	5
2.4 Q-Dir	6
2.5 Nautilus	7
<b>3 Approach</b>	<b>10</b>
3.1 Background	10
3.1.1 File system	10
3.1.2 Main architectural concepts	11
3.2 Tools	12
3.3 Application architecture	12
<b>4 Main features</b>	<b>14</b>
4.1 File operations	14
4.1.1 Open	14
4.1.2 Create	14
4.1.3 Copy	15
4.1.4 Cut	15
4.1.5 Paste	15
4.1.6 Delete	16
4.1.7 Rename	16
4.1.8 Shortcuts	16
4.1.9 Properties	17
4.2 Filtering	18
4.3 Sorting	18
4.4 Navigation line	19
4.5 Global search	19
4.6 Favorite paths	20
4.7 Cloud drives support	21
4.8 Information panel	22

<b>5</b>	<b>UI/UX Design</b>	<b>25</b>
5.1	General overview . . . . .	25
5.2	User flow . . . . .	31
5.3	Themes . . . . .	32
5.3.1	Light theme . . . . .	33
5.3.2	Dark theme . . . . .	33
<b>6</b>	<b>Evaluation</b>	<b>35</b>
6.1	Testing environment . . . . .	35
6.2	Results . . . . .	35
<b>7</b>	<b>Summary</b>	<b>37</b>
7.1	Conclusion . . . . .	37
7.2	Future work . . . . .	37
<b>A</b>	<b>References</b>	<b>38</b>
	<b>Bibliography</b>	<b>39</b>

# List of Figures

2.1	Total Commander	2
2.2	Directory Opus	3
2.3	Windows Explorer	5
2.4	Q-Dir	6
2.5	Nautilus	7
3.1	File systems comparison	11
3.2	MVC architecture	11
3.3	QExplorer UML Diagram	13
4.1	Properties window	17
4.2	Sorting header	18
4.3	Favorites widget life cycle	21
4.4	Cloud storages market growth	21
5.1	First application prototype	25
5.2	Final application prototype	26
5.3	One file, one folder, multiselection and blank space context menus from left to right	27
5.4	Favorite paths window	27
5.5	Navigation line	28
5.6	Combobox for swapping drives	28
5.7	"Edit" drop-down menu	28
5.8	Global search window	29
5.9	"Options" drop-down menu	29
5.10	Two modes of cloud storage widget	29
5.11	Total memory chart	30
5.12	Types distribution chart	30
5.13	Drives information widget	31
5.14		32
5.15	Light theme	33
5.16	Dark theme	33
6.1	Performance metrics	35



# List of Tables

2.1	Pros and cons of Total Commander	3
2.2	Pros and cons of Directory Opus	4
2.3	Pros and cons of Windows Explorer	6
2.4	Pros and cons of Q-Dir	7
2.5	Pros and cons of Nautilus	8
2.6	File managers comparison	9
3.1	Used tools	12
4.1	File types and extensions mapping	23
6.1	Code metrics #1	36
6.2	Code metrics #2	36

# List of Abbreviations

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

*Dedicated to my loving parents*

## Chapter 1

# Introduction

### 1.1 Motivation

A file manager is a computer program that provides a user interface for working with a file system and files. File manager allows you to perform the most common operations on files - creating, opening, editing, moving, renaming, copying, deleting, changing attributes and properties, searching for files and assigning rights. In addition to the basic functions, many file managers include a number of additional features, such as networking, backup, printer management, etc. They are often accompanied by additional utilities that make life easier for the user. For many users, the favorite file manager often acts as a shell, replacing some of the standard file management tools available in the operating system.

However, for people who have experience in this area, it is obvious that today's file managers are far from ideal. Some do not have enough functionality, which is why some specialists cannot use them. Standard solutions provided by operating system developers, such as Windows Explorer or Nautilus for Ubuntu, immediately come to mind. There is also another extreme, oversaturation with functionality, which affects other characteristics, such as ease of use. An example is Total Commander, a favorite manager for many people. However, when someone installs it, they immediately get lost in many buttons and features that they will never use. A critical shortcoming of some managers is UI/UX design problems. The developers are trying to put in their project everything that competitors have, but do not notice that their product is almost impossible to use. In terms of user experience, it stuck in the nineties.

### 1.2 Task definition

So I want to create my own file manager, in which I will try to neutralize all the shortcomings that I have repeatedly encountered.

#### **Requirements:**

- Fast
- Convenient
- Adaptive
- Optimal functionality
- Cross-platform

## Chapter 2

# Existing solutions

### 2.1 Total Commander

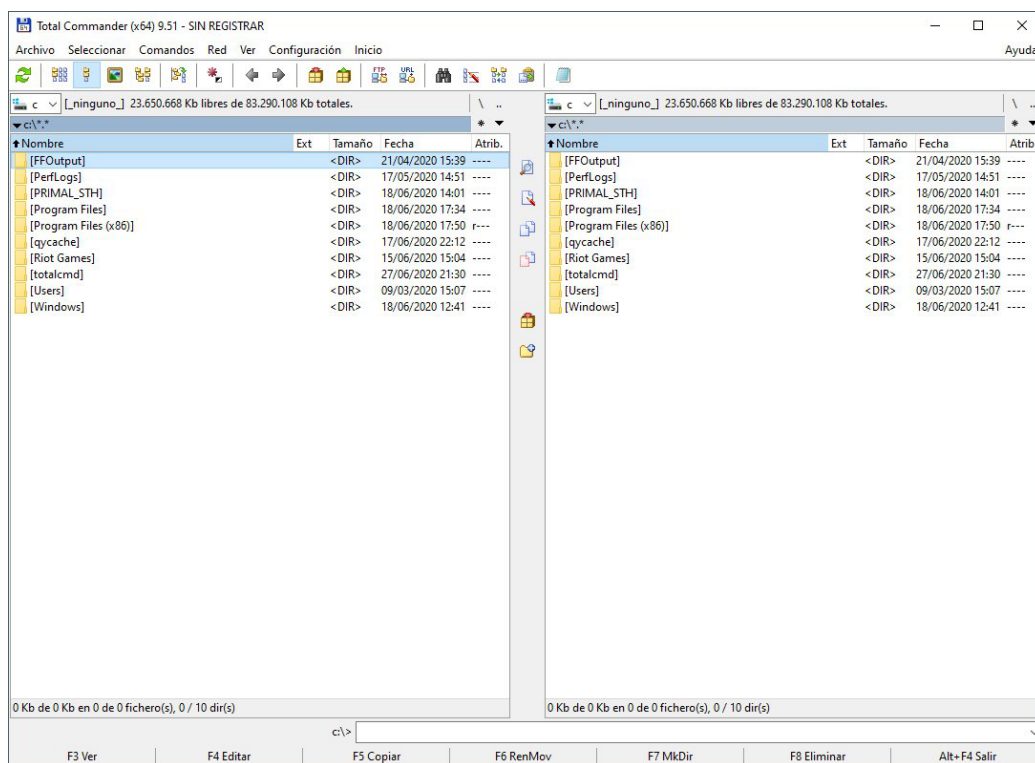


FIGURE 2.1: Total Commander

Total Commander is a two-pane file manager for 32 and 64-bit Windows operating systems that allows you not only to simplify routine file operations (copy, paste, delete, move, etc.), but also has a lot of additional useful options.

#### Main features:

- Show/select files with specific search pattern, size, date or contents.
- Enhanced search function with full text search in any files across multiple drives.
- Command line for starting of programs with parameters.
- Split/Combine big files.
- Custom columns view to show additional file details.

- Compare files by content.
- Search for duplicate files.
- HTML- and Unicode-Viewer in Lister.
- Built-in FTP client supports most public FTP servers, and some mainframes.
- Configurable main menu.
- Archive handling.
- Parallel port transfer function (direct cable connection).

### Total commander general features

Pros	Cons
Plugins support	Need to spend a lot of time for custom configuration
Very stable	Hard to get used to
Allows to work without a mouse	
Customizable	
Multifunctional	

TABLE 2.1: Pros and cons of Total Commander

## 2.2 Directory Opus

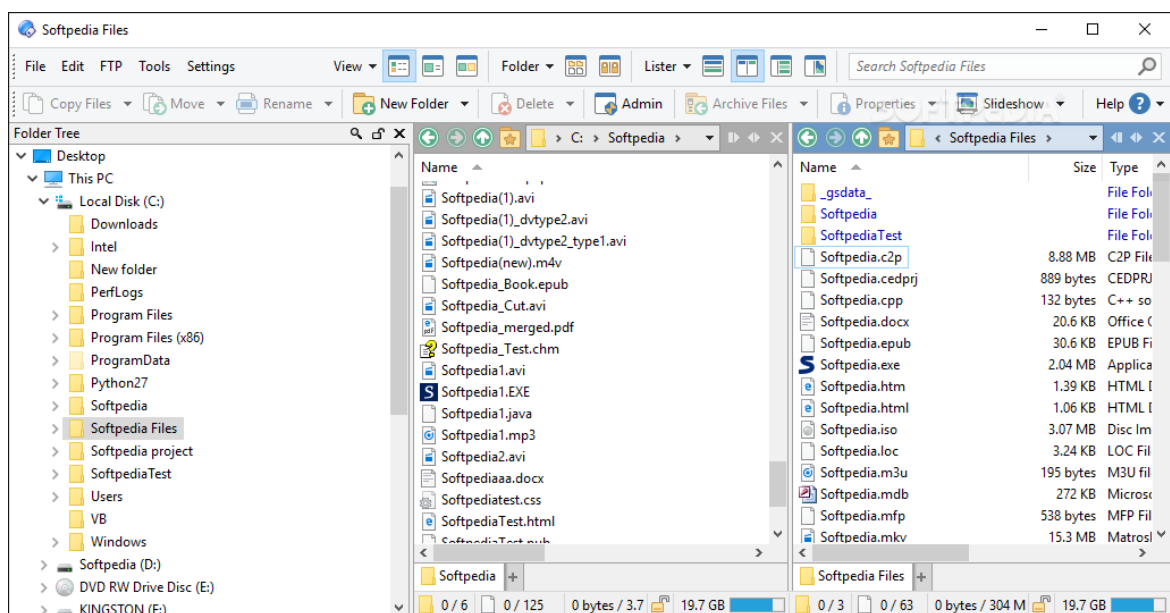


FIGURE 2.2: Directory Opus

Directory Opus is a powerful and easy-to-use file manager that provides all basic file operations. The capabilities of the manager don't differ in something special in

comparison with similar programs. Developers point out that Directory Opus was conceived mainly as a replacement for the standard Windows Explorer. Thus, this application will facilitate the work with files, making it more convenient and comfortable. Directory Opus supports working with archives, provides FTP access, processes music and graphics files, syncs with OneDrive cloud storage, and much more.

### Main features:

- Single or dual file displays, with single or dual trees.
- Folder tabs let keep multiple folders open and switch quickly between them.
- Integrated viewer pane to preview many common image and document file formats.
- View and edit file metadata (EXIF, MP3, PDF, etc).
- Sorting, grouping, filtering and searching.
- Support for FTP, Zip, 7-Zip, RAR and many other archive formats.
- Access content on portable devices like phones, tablets and cameras.
- Built-in tools including synchronize, duplicate file finder, image converter and uploader.
- Print or export folder listings, copy file listings to the clipboard, calculate folder sizes.
- Queue multiple file copies for improved performance.
- Support for CD/DVD burning.
- Fully configurable user interface - colors, fonts, toolbars, keyboard hotkeys.
- Full scripting interface supports VBScript, JScript or any compatible installed Active Scripting language.

### *Directory opus general features*

Pros	Cons
Customizable UI	Intimidating to inexperienced users
Tabs	Takes a lot of memory
Ongoing application maintenance and customer support	Expensive

TABLE 2.2: Pros and cons of Directory Opus

## 2.3 Windows Explorer

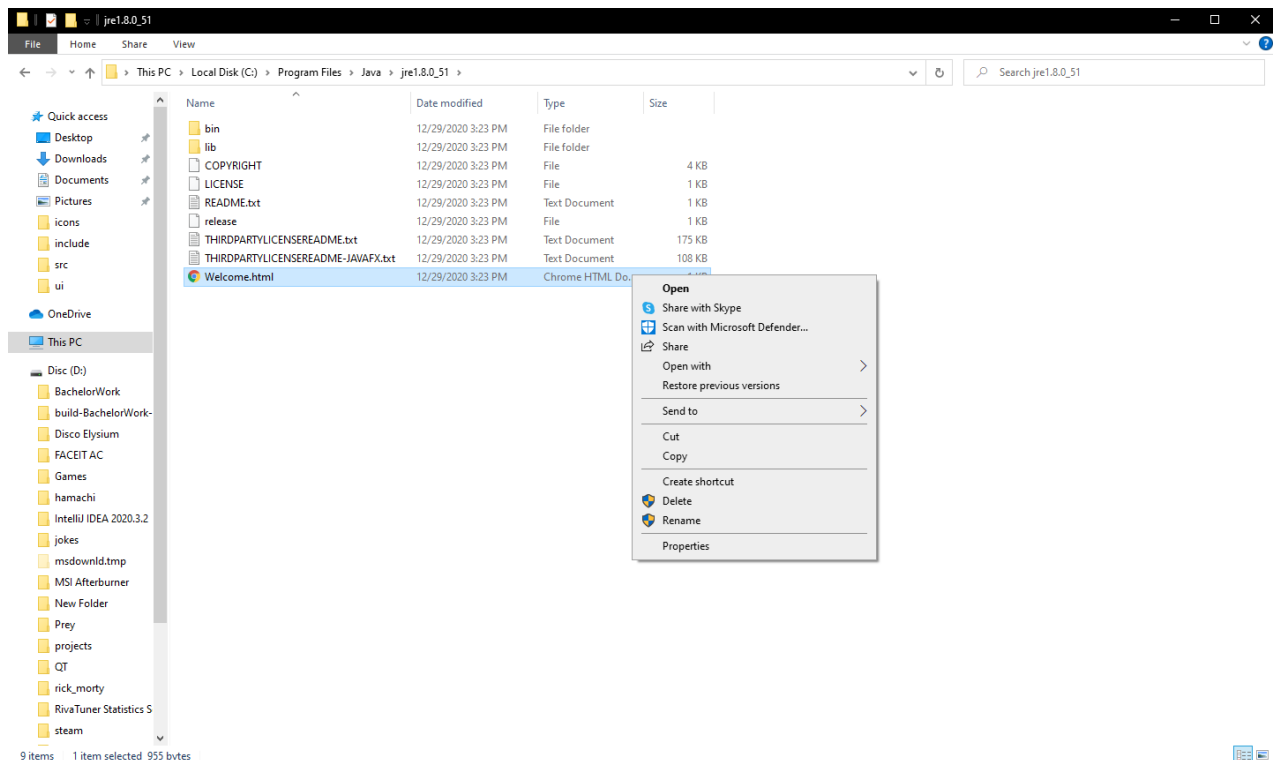


FIGURE 2.3: Windows Explorer

Windows Explorer is an application that implements a graphical user interface for accessing files in the Microsoft Windows operating system. Explorer is now the de facto mainstay of the graphical user interface for Windows. Everything that the user sees after Windows boots - desktop icons, taskbar, start menu is Windows Explorer.

### Main features:

- Access frequently used commands.
- Navigate directly to a different location, including local and network disks, folders, and web locations..
- Perform instant searches, which show only those files that match what you typed in the Search box for the current folder and any of its subfolders.
- Display common folders, such as Favorites, SkyDrive, Homegroup (a shared network), This PC, and Network.
- Extended file information (metadata).
- Customizable interface.
- Cloud drive support.
- Drives cleanup.
- Optimize and defragment drives.



*Windows Explorer general features*

Pros	Cons
Preinstalled on Windows	Overly simplistic
Good choice for users with low requirements	Little functionality
	Unconfigurable

TABLE 2.3: Pros and cons of Windows Explorer

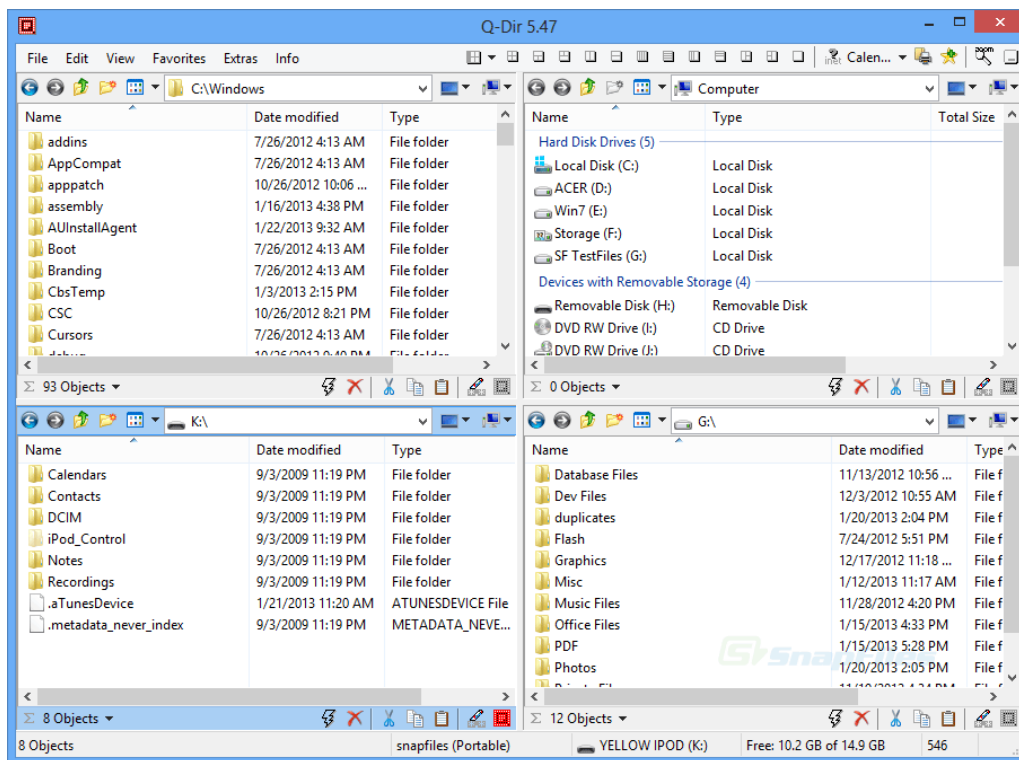
**2.4 Q-Dir**

FIGURE 2.4: Q-Dir

Q-Dir is a file manager with a four-pane interface, support for zip archives, FTP and highlighting folders and files with different extensions. By default, Q-Dir uses a four-pane interface (4 windows), but provides the ability to customize the interface, allowing you to specify 3 or 2 panels with a vertical or horizontal arrangement. Q-Dir is tightly integrated with Windows Explorer. For example, the standard panel views are used. Lists, tables, thumbnails - everything is taken from the standard Windows tool.

**Main features:**

- File management in 4-window with tabs.
- Folder size with extra information.

- Color filter for files and folders.
- Directory structure with visible tree branches.
- Based on the MS Windows OS File Manager.
- Full Unicode Support.
- Save folder combinations as favorites.
- Mark selected folders and files.
- Improved quad explorer file preview.
- Multilingual.
- Low System Resource usage.

### *Q-Dir general features*

Pros	Cons
Lightweight	Some bugs
Tabs	Some critical features such as search are missing
Customizable	
Free	

TABLE 2.4: Pros and cons of Q-Dir

## 2.5 Nautilus

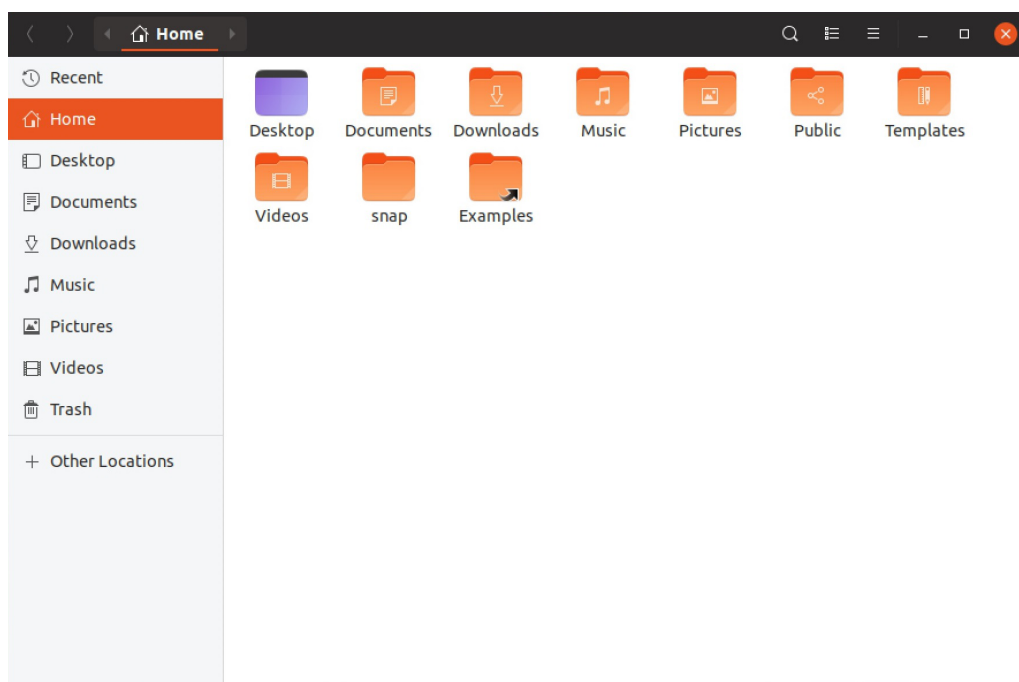


FIGURE 2.5: Nautilus

Nautilus is a file manager for the GNOME and Unity desktop environments. In Ubuntu, it is installed as the main file manager as well as the desktop manager.

### Main features:

- Connect and disconnect storage devices (hard drives, network drives, flash drives, optical drives, etc.).
- Work with remote servers (FTP, SSH, WebDAV, SMB).
- View thumbnails of files (videos, images, PDF, DJVU, text files).
- View properties of files and directories (including additional properties on separate tabs using third-party applications).
- Create, modify, delete files and directories (including using file templates located in the `/ Templates` or `/ Templates` directory).
- Run scripts and applications.
- Search files and directories by name.
- Place files and directories on the desktop.
- Burn CD / DVD discs (using Brasero).

### *Nautilus general features*

Pros	Cons
Easy to use	Very little functionality
Minimalistic	Unconfigurable
Supported on all Linux distributions	Memory leaks
	Many bugs

TABLE 2.5: Pros and cons of Nautilus

File manager	Total Commander	Comman-der	Directory Opus	Explorer	Q-Dir	Nautilus	qExplorer
Supported system	Windows, Windows CE, Android	Windows	AmigaOS, Windows	Windows	Windows	Linux	Windows, Linux, macOS
Search for files using regular expressions	+		+	-	+	+	+
FTP client	+		+	+	-	+	-
Text search in files	+		+	-	-	-	+
Multiselection	+		+	+	+	+	+
Archive handling	-		+	+	+	+	-
Configurable user interface	+		+	+	+	-	+
Saving favorites folders	+		-	+	+	-	+
Multilingual	+		+	+	+	+	-
Cloud storage support	-		-	+	-	-	+
System info panel	-		-	-	-	-	+
Global file search	+		+	+	+	+	+
Multiple panes	+		+	-	+	-	+
Built-in command line	+		-	-	-	-	-

TABLE 2.6: File managers comparison

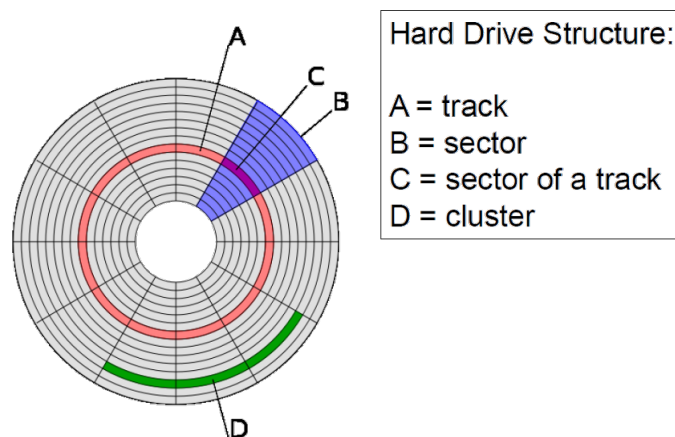
## Chapter 3

# Approach

### 3.1 Background

#### 3.1.1 File system

File system - the order that determines the method of organization, storage and naming of data on media in computers, as well as in other electronic equipment. The file system determines the format of the content and physical storage of information, which is grouped as files. The file system connects the storage medium on the one hand and the file access API on the other. When an application accesses a file, it has no idea how the information is located in a particular file, as well as on what physical type of media (CD, hard disk, magnetic tape, flash memory block) it is recorded. All the program knows is the file name, its size and attributes. It receives this data from the file system driver. It is the file system that determines where and how a file will be written to physical media (such as a hard drive).



In terms of the operating system, the entire disk is a set of clusters (usually 512 bytes or larger). File system drivers organize clusters into files and directories (which are actually files that contain a list of files in that directory). The same drivers keep track of which of the clusters are currently in use, which are free, and which are marked as faulty. *File systems*. In addition to user files, the file system also contains its own parameters (such as block size), file descriptors (file size, location, snippets, etc.), file names, and a directory hierarchy. It can also store security information, advanced attributes, and other settings. Basically, it depends on the type of file system, which there are a lot of even in terms of single operating system.

	LINUX COMPATIBLE	WINDOWS COMPATIBLE	MACOS COMPATIBLE	FEATURES
FAT32	✓	✓	✓	FS < 4GB 2 TB max
EXT2	✓	✗	✗	FS < 2TB 32 TB max
EXT3	✓	✗	✗	Journaling system
EXT4	✓	✗	✗	FS < 16TB 1 EB max
NTFS	✓	✓	✓	FS < 16EB* 16EB max
HFS	✓	✗	✓	FS < 8EB 8EB max

FIGURE 3.1: File systems comparison

### 3.1.2 Main architectural concepts

The program is built according to the MVC pattern. It is divided into three separate parts. Model is responsible for storing data and its structure. View is responsible for presenting this data to the user, ie program interface. Controller controls components, receives signals in response to user actions (changing the position of the mouse cursor, pressing a button, entering data into a text box) and transmits data to the model.

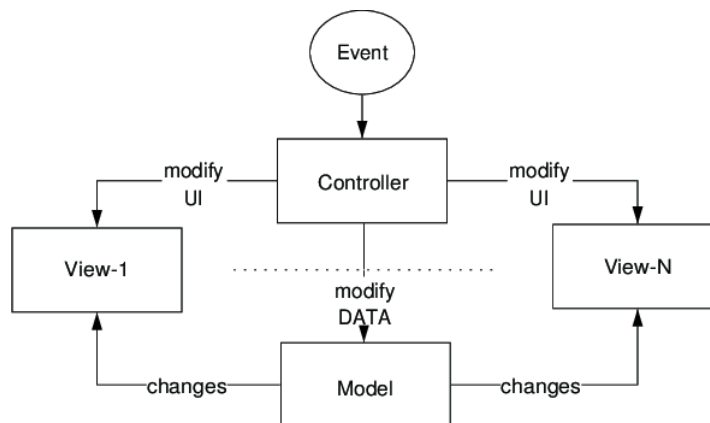


FIGURE 3.2: MVC architecture

Role of model is played by different business logic and file system representation. In order to get software access to the file system, Qt library was used, namely the `QFileSystemModel` class. This class provides access to the local filesystem and different ways to interact with it. It is quite well optimized in terms of performance. It uses `QFileSystemWatcher` to automatically keep all the information up to date. *Official Qt documentation. QFileSystem.* View part is represented with Qt widgets structure described in ui forms. All of these widgets inherit `QWidget` class and can be used to display different data, receive user input and also serve as a container for other widgets. As mentioned above view is connected to model with controller. In this case it is signals. They are triggered when user performs specific action. Each signal has some slot (usual C++ function) connected to it. When signal is triggered, corresponding slot is executed.

## 3.2 Tools

	Name	Version	Purpose
<b>Programming language</b>	C++	11	Serves to provide basic syntax and define program behavior.
<b>Project assembly automation system</b>	CMake	3.19.2	Used to automate building software from source code. Checks for the presence of the necessary libraries and includes them, builds projects under different compilers and operating systems.
<b>Compiler</b>	MinGW g++	8.1.0	Converts source code written on C++ into semantically equivalent machine code that is required to run a program by a computer.
<b>Software development framework</b>	Qt	5.15.2	One of the most convenient frameworks for GUI application development. Qt provides not only suitable set of class libraries, but also a specific application development model. An important advantage of Qt is its well thought out and logical set of classes, which provides high level of abstraction. In addition, Qt is cross-platform and in order to run the program on another OS, you just need to recompile the source code.
<b>Library for building charts</b>	Qt Charts	5.15.2	Provides a set of easy to use chart components. It uses the Qt Graphics View Framework, therefore charts can be easily integrated to user interfaces.
<b>IDE for writing code</b>	Qt Creator	4.14.0	The best development environment for Qt. Has native ui forms support as well as built-in UI Building Tool.
<b>IDE for creating ui forms</b>	Qt Designer	4.14.0	Used to create and configure Qt designer forms(*.ui) These forms represent a tree of widgets in XML format(QML) and will be converted to C ++ code that can be compiled. This way you can much more flexibly customize the appearance of the widget when it has a complex structure and it makes no sense to build it directly in the code.

TABLE 3.1: Used tools

## 3.3 Application architecture





## Chapter 4

# Main features

### 4.1 File operations

#### 4.1.1 Open

- **Description**

Ability to open different items in the file system. To open a file you need to double-click on it or select the appropriate option in the context menu. Afterwards the file will be opened using a default program. To open a folder, double-click it. Current directory in corresponding panel will be updated.

- **Purpose of use**

Reading and writing files as well as navigating through the file system tree is one of the most important functions of any file manager.

- **Realisation**

Opening files is based on `bool QDesktopServices::openUrl(const QUrl &url)` function. We get file absolute path using `QString QFileInfo::absoluteFilePath()` and insert it as an argument. This action is connect to double-clicking and corresponding context menu option. Folder opening is realised by default when connecting `QTableView` and `QFileSystemModel`.

#### 4.1.2 Create

- **Description**

Creating files and folders. To do this, select the appropriate option in the context menu, then enter the name. The item will appear in the current directory. Thanks to multiselection, it has become possible to create a folder with predefined content. To do this, select a few items, call the context menu and create a folder. Once created, it will contain all the selected items.

- **Purpose of use**

Creating new nodes and elements is one of the important functions of the file system, which also needs to be managed.

- **Realisation**

Creation of files is realised using

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

After calling this function Qt will create file which does not exist for now and at the end it will be automatically closed. In case if such file exists certain custom behavior is established. File creator will try to add some counter at the end of the file name in the cycle, so the following file names can be produced: filename(1).txt, filename(2).txt, etc... The same works for folders, but `bool QDir::mkpath(const QString &dirPath)` is used instead of open function for files.

### 4.1.3 Copy

- **Description**

Marks an item as to be copied. This is needed to know exactly what needs to be pasted in future. Multiselection is supported.

- **Purpose of use**

Often the user may need a duplicate of folder or file. For example, to try or test something. The original version must be preserved in case of a negative result.

- **Realisation**

Since it's just marking, there is no complicated logic. All selected entries are placed in the global buffer, from which they will be taken in case of pasting.

### 4.1.4 Cut

- **Description**

Cutting is the same as copying, except that the source is not saved. After cutting, we will have only one copy of the item, but in the target directory instead of the original. Multiselection is supported.

- **Purpose of use**

Cutting is actually a directory change. This way, user can move the item through the file system tree.

- **Realisation**

The same thing happens as in case of copying with one difference. At the end of this process, the global variable `bool toCut` is assigned a value `true`.

### 4.1.5 Paste

- **Description**

Pasting is directly related to copying and cutting. While in the previous step we marked file/folder as to be copied/cut, now we execute this operation, specifying the target directory. Only then items will appear there.

- **Purpose of use**

To choose target directory and confirm cut/copy operation execution.

- **Realisation**

In the case of files, the following function

```
bool QFile::copy(const QString &fileName, const QString &newName)
```

is used. It copies file from source location(1st argument) to destination location(2nd argument). With folders, everything is a little harder. It is necessary

to iterate recursively over initial directory twice. During the first time, we create the same directory structure in the target location as in the original one. For more details about creation, see 4.1.2 During the second time - copy all the files in turn to the appropriate locations created during the previous iteration. After that, global variable `bool toCut` is checked. In case of `true`, file/folder is deleted from destination location. For more details about deletion, see 4.1.6

#### 4.1.6 Delete

- **Description**  
Remove files or entire folders from the file system. After that, the memory they occupied on the hard drive will be freed. Multiselection is supported.
- **Purpose of use**  
Clear the memory occupied by unnecessary files
- **Realisation**  
Various access and rights checks occur before deletion begins. If these checks fail, corresponding warning will be displayed. In case of success, file/folder will be deleted. The removal functions are provided by Qt by default. For files it is `bool QFile::remove()`, folders - `bool QDir::removeRecursively()`.

#### 4.1.7 Rename

- **Description**  
File name is a string of characters that uniquely identifies a file/folder in directory. File names are built according to the rules adopted in a particular file and operating system. File names are also used in absolute/relative path building so you can access it everywhere. This feature allows to change such identifier.
- **Purpose of use**  
Renaming files is important to better understand what each file is. This way, you can roughly evaluate its content without even opening it.
- **Realisation**  
Before renaming a file, it is checked whether the user can edit it:  
`info.permission(QFile::WriteUser)` After passing this check, the file is renamed using the functions:

```
bool QFile::rename(const QString &newName);  
bool QDir::rename(const QString &oldName, const QString &newName);
```

Case when such file/folder already exists is dealt the same way as for creation 4.1.2.

#### 4.1.8 Shortcuts

- **Description**  
In terms of file system, shortcut is a handle that allows the user to find a file or resource located in a different directory or folder from the place where the shortcut is located. *Shortcuts* Feature allows to create shortcuts for files and folders. When you open the file shortcut, the corresponding file will open, when you open the folder shortcut, current directory in the active panel will change.

- **Purpose of use**

Sometimes file can be strongly tied to its directory and its location cannot be changed. However, the user often accesses this file and would like to do so from a convenient location without changing the location of the file. Shortcuts are needed to provide this capability. They provide quick access to any element of the file system.

- **Realisation**

Creation of shortcuts is based on `bool QFile::link(const QString &linkName)` function. When user wants to create a shortcut for a specific file, this function is called. Arguments are the absolute path of the file and the same path but with `.lnk` extension, which is actually an extension of the shortcuts. For folders, the process is identical.

### 4.1.9 Properties

- **Description**

Window with detailed information about chosen item. This information contains the following data: file name, type(file, directory symbolic link, etc), size, parent folder, group(Unix systems), owner(Unix systems), date when this item was created, last modified date.

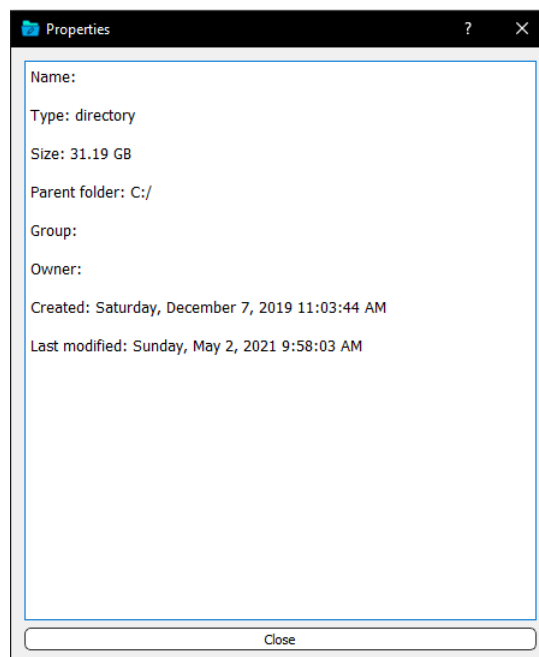


FIGURE 4.1: Properties window

- **Purpose of use**

User may need some additional information about file. Things like size, absolute path and type can be extremely useful to find out.

- **Realisation**

Usual popup window, filled with content from `QFileInfo` class.

## 4.2 Filtering

- **Description**

Filtering in the application is implemented using a special field. In this field you can enter both the full file name and the template by which you want to search for it. Regular expressions are supported.

- **Purpose of use**

Quick and easy replacement for global search, but with some limitations. Saves time when you need to find a file by its name within a single directory.

- **Realisation**

When the user confirms his input, a certain custom function is triggered. Based on this input, a request is formed, after which the appropriate filters are applied to the file system model.

```
QStringList filters;
QString request = "*";
request.append(item_name);
request.append("*");
filters << request;
filters << ".";
filters << "..";
model->setNameFilters(filters);
```

## 4.3 Sorting

- **Description**

Sorting in the project is implemented using a header located on top of each panel. You can click on the category of interest and sort all the entries in ascending/descending order.

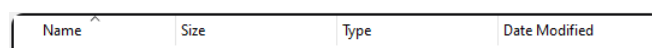


FIGURE 4.2: Sorting header

- **Purpose of use**

Sorting may be useful in many situations. For example, find the largest/latest modified/oldest, etc. file in the folder.

- **Realisation**

To represent the file system `QTableView` is used. By default it has header which is actually `QHeaderView` class. To be able to use it for sorting, only the basic configuration is needed.

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

---

## 4.4 Navigation line

- **Description**

The navigation line serves to display and control the user's location in the file system tree. It contains the absolute path of current directory. Each time it changes, the line text is also updated. In addition, you can enter your path and if it is valid, then by pressing the appropriate button, you will go to the desired directory. Each panel has its own navigation line.

- **Purpose of use**

Used to display the absolute path of the current directory, as well as quickly go to the user-specified path.

- **Realisation**

Usual `QLineEdit` widget and there is no complicated logic behind it. When user clicks enter on keyboard, input is checked for validity and there happens a transition to it. In addition, whenever the directory is changed, the new absolute path is written to the appropriate navigation line.

## 4.5 Global search

- **Description**

Allows you to search for a file throughout the file system according to the following criteria: file name, containing text, parent directory. As the search engine finds the files, they will appear in a list where they can be opened. Does not support binaries and archives.

- **Purpose of use**

A very powerful mechanism that makes it easy to find anything in the file system. It is enough for the user to know at least a certain set of characters and most likely the search will be successful if there are not many such files.

- **Realisation**

After user confirms his input, data received from three fields is validated and processed. In case if path field is empty, root directory is used. If file name is not specified - the search engine does not take it into account and file with any name is the search target. The same thing happens when text field is not filled. To iterate over directories in order to find searched files `QDirIterator` is used:

```
QDirIterator it(path, filter, QDir::AllEntries | QDir::NoSymLinks
    | QDir::NoDotAndDotDot, QDirIterator::Subdirectories);
```

Afterwards search function is started in the new thread using `std::thread` from C++ concurrent library. The purpose is to not stop main application flow with such a long-term process. Files will appear in the window as they are found and user can interact with them even if search process has not finished yet. As mentioned above, iteration over directories is being executed. In case if user wants to search for containing text some additional operations are needed.

If the file fits all the parameters, engine must also check it for containing text. To do so file should be opened and scanned with `QTextStream`:

```
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];
            break;
        }
    }
}
```

Once all the files have been found or the window has been closed, search process finishes.

## 4.6 Favorite paths

- **Description**

Feature to save and go to frequently used directories. If desired, user can save frequently visited directories, edit them, and delete them if they are no longer needed.

- **Purpose of use**

Often important folders can be located very deeply and therefore inconvenient to access. To simplify the user's life and save his time this feature has been implemented. The main advantage over shortcuts is that the list of favorite paths is available anywhere, because it is not tied to the file system. Thus, user has not to go to the folder with shortcuts, but simply press one button.

- **Realisation**

The main role in the implementation of this feature is played by two basic widgets: `FavoritePathWidget` and `FavoritePathWidgetContainer`. As can be seen from their names, first one corresponds to a favorite path and can be added by the user, the other serves as a container for paths and can be called with the appropriate button. When user tries to add a new path, first of all it passes simple validation.

```
if (QDir(path).exists() && !path.isEmpty()) {
    return true;
}
return false;
```

After successful validation, the widget is created and added to the container. Slot with opening of the corresponding folder is connected to this widget. In order to preserve path even after application is closed, it must be added to some external storage. In our case it is simply a config file in which string representation of a path is written. In addition, there is a local cache, which loads all the entries in order to not read them from the file every time. This

way, all interactions go through this cache and only after closing the widget, changes are applied to the main storage.

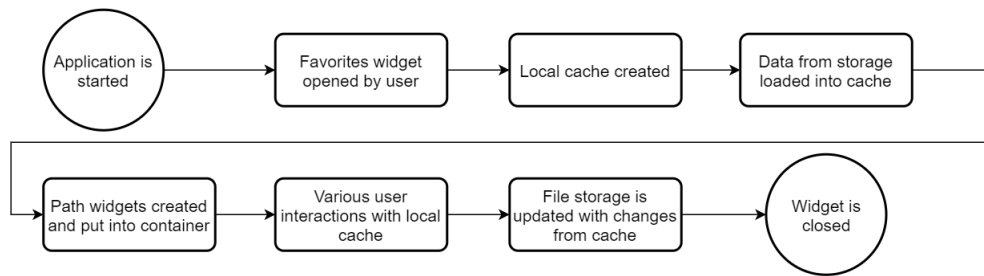


FIGURE 4.3: Favorites widget life cycle

From ui point of view there are no limitations on the amount of entries as they are located in `QScrollArea` and after they can no longer be displayed within the container, a slider will be added automatically.

## 4.7 Cloud drives support

- **Description**

Widget that allows you to quickly open your cloud storage, as well as provides additional information about it: username, used memory, types distribution. Supports not only OneDrive but any other storage as long as it has local representation on the device.

- **Purpose of use**

Cloud storage is a model of storing data in a computer, in which digital data is stored in logical pools, and physical storage covers several servers (usually in several places). Cloud storage gives users instant access to a wide range of resources and applications hosted in another organization's infrastructure through a web service interface. The main advantage is that it is not tied to something local, you can use it from any device anywhere. Recently, the cloud storage market has shown tremendous growth. *Cloud storage*

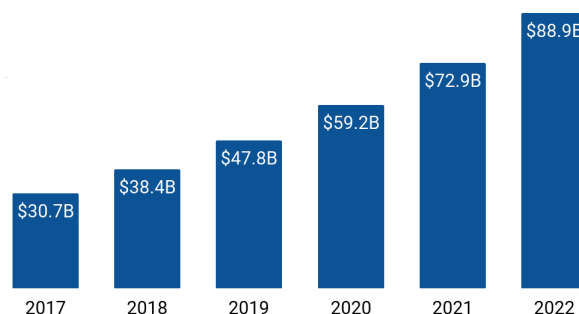


FIGURE 4.4: Cloud storages market growth

- **Realisation**

As mentioned above, each cloud storage has its own local representation on



the user's device. They are usually located in the home folder. To get cross-platform access to this location: `QString QDir::homePath()`. Afterwards it begins widget setup. We create click signal and emit it in case of `void CloudDriveWidget::mousePressEvent(QMouseEvent *event)`. Opening a local disk folder connects to this signal. Also, additional information is extracted using the `QFileInfo` class, as well as the type distribution mechanism. For more information, see 3. Having this information, we can fill the widget with it, after what it is ready to use.

## 4.8 Information panel

Information panel, located in right side of the interface displays different information about file system, directory where user is located, existing drives, etc. It contains three widgets:

### 1. Total memory distribution chart

- **Description**

Pie chart is divided into two sections: outer and inner. The inner one shows how memory is distributed between virtual disks. The outer one shows the distribution between occupied and free memory for each of the disks. For more information from ui point of view, see 11

- **Purpose of use**

The user should always be aware of the state of his memory. This way, it can be determined which disk is the highest priority to use, how large files it can store, and when to clear the memory for future use.

- **Realisation**

The Qt Charts library was used as a basis for building charts. Information about memory state was taken from `QStorageInfo` class:

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

Having this information, total memory distribution chart can be easily built:

```
DonutBreakdownChart chart = new DonutBreakdownChart();
for (QStorageInfo info : infoList) {
    QPieSeries *series = new QPieSeries();
    series->setName(info.rootPath());
    series->append("Free", info.bytesAvailable());
    series->append("Used", info.bytesTotal() - info.
bytesAvailable());
    donutBreakdown->addBreakdownSeries(series);
}
```

### 2. Drives information widget

- **Description**

The widget shows more detailed information about each of the disks

present in the operating system, including name, capacity, file system type, whether the disk is mounted and whether it is read-only. For more information from ui point of view, see [13](#)

- **Purpose of use**

There are many use cases where it could be useful. Unmounted discs cannot be used until they are re-mounted manually. Read-only discs cannot accept any changes. Many factors also depend on the type of file system.

- **Realisation**

Drives information widget is represented by `QTabWidget` with `n` tabs, where `n` is equal to number of drives. To get information about all drives: `QList<QStorageInfo> QStorageInfo::mountedVolumes()`. After that, we iterate over this list, extract the required information from each entry(drive) and fill widget with it.

### 3. Types distribution chart

- **Description**

Pie chart has dynamic number of slices. Each of these slices represents a specific file type.

Type	Extensions
<b>Audio</b>	aif, cda, mid, midi, mp3, mpa, ogg, wav, wma, wpl
<b>Archive</b>	7z, arj, deb, pkg, rar, rpm, gz, z, zip
<b>Data</b>	csv, dat, db, log, mdb, sav, sql, xml
<b>Executable</b>	apk, bat, bin, com, exe, gadget, jar, msi, wsf
<b>Image</b>	ai, bmp, gif, ico, jpeg, jpg, png, ps, psd, svg, tif, tiff
<b>Programming file</b>	c, cgi, pl, class, cpp, cs, h, java, php, py, sh, swift, vb
<b>System</b>	bak, cab, cfg, cpl, cur, dll, dmp, drv, icns, ico, ini, lnk, msi, sys, tmp
<b>Video</b>	3g2, 3gp, avi, flv, h264, m4v, mkv, mov, mp4, mpg, mpeg, rm, swf, vob, wmv
<b>Text</b>	doc, docx, odt, pdf, rtf, tex, txt, wpd

TABLE 4.1: File types and extensions mapping

This way, it can be estimated type distribution in the current directory. When the directory is changed, the chart is updated. For more information from ui point of view, see [12](#)

- **Purpose of use**

Can be useful to characterize a directory in terms of existing file types, check for extra types, and quickly understand what it is used for.

- **Realisation**

The Qt Charts library was used as a basis for building charts. First of all we get all the files in current directory using

```
QFileInfoList QDir::entryInfoList(QDir::Filters filters);
```

After that we iterate over this list, define type of each file and then increase the counter of the corresponding entry in `QMap <QString, int>`.

Having this distribution, QChart can be easily built.

```
QPieSeries *series = new QPieSeries();
QMapIterator<QString, int> it(typesDistributionMap);
while (it.hasNext()) {
    it.next();
    series->append(it.key(), it.value());
}
QChart typesChart = new QChart();
typesChart->addSeries(series);
```

# Chapter 5

# UI/UX Design

## 5.1 General overview

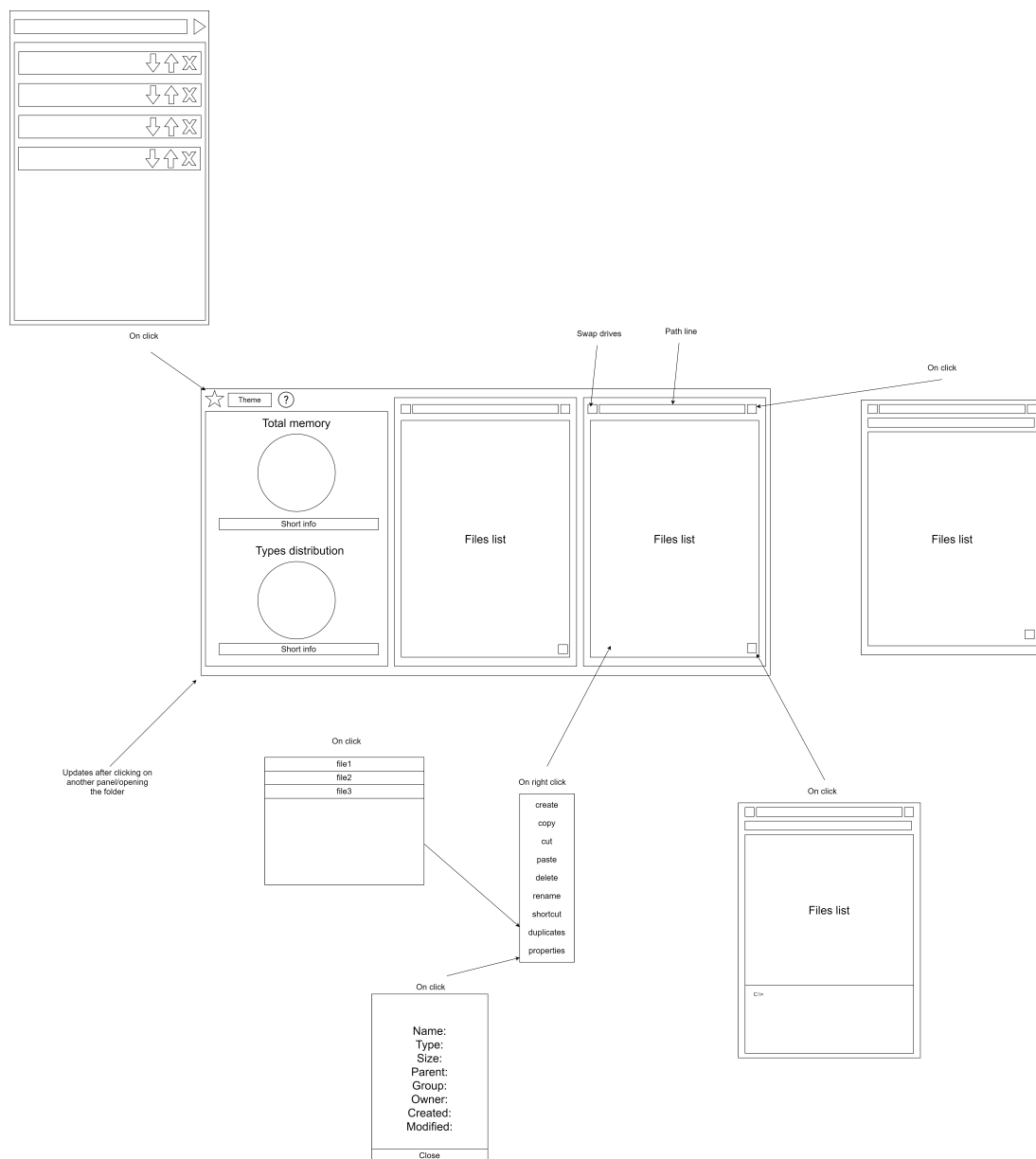


FIGURE 5.1: First application prototype

The figure above shows the first prototype of the planned application. Many details have been preserved in the final version, although of course some have been reconsidered, some removed, and some things added. For example, the set of buttons for controlling your favorite paths has changed, the information panel has been shifted to the right, and new widgets, such as cloud storage, have been added to it. In addition, the overall appearance of the application has changed slightly. Now let's take a look at the final version of the design.

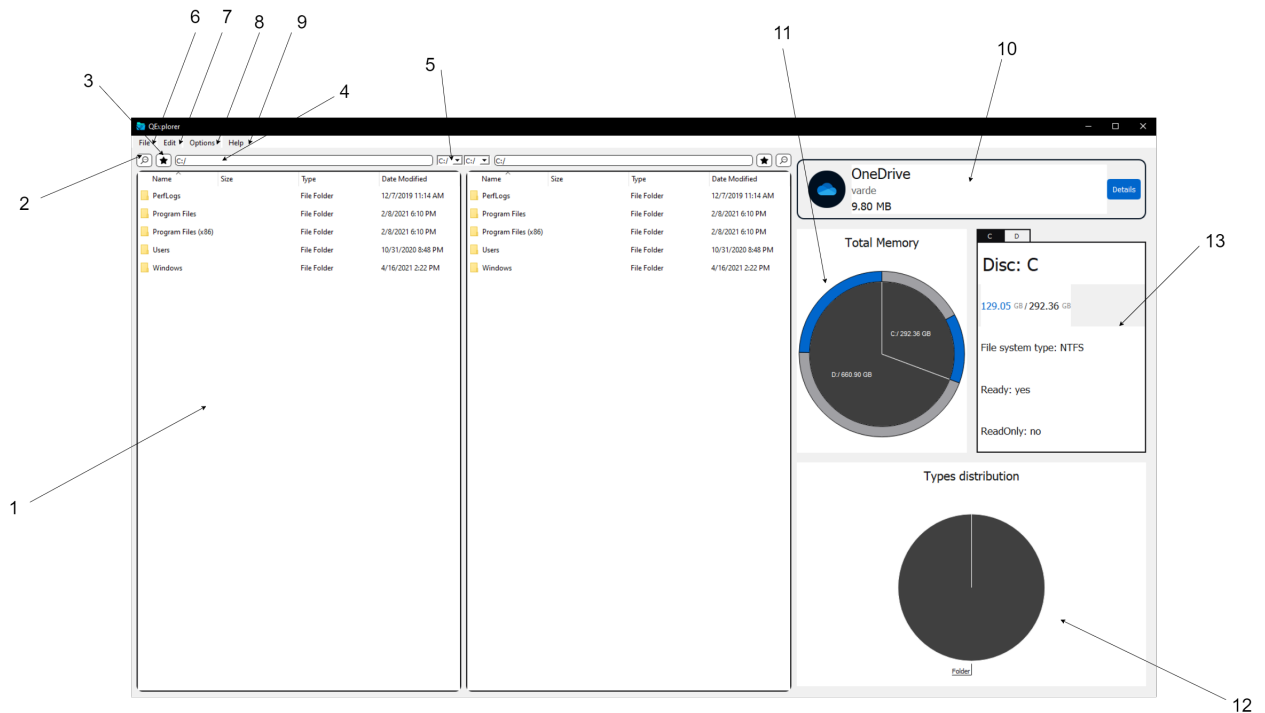


FIGURE 5.2: Final application prototype

### 1. Files list view

This list is the place where all files and folders are displayed. The elements are arranged one after another sorted in alphabetic order by default. A certain set of additional information is shown for each element. This is the icon assigned to the file in the operating system, name, type, size and date of the last modification. You can use header section to sort by each of the columns. For detailed information about sorting, see ??

User can interact with files list in many ways. The main purpose is navigation. We can move from one folder to another, thus moving through the file system. If you double-click a file, it will be opened with the standard program assigned to it. For some additional operations you need to call the context menu. This can be done by right-clicking anywhere in the list. Depending on the location of the click, a menu will open with a specific set of actions shown on the figure below. For detailed information about file operations, see [4.1](#)

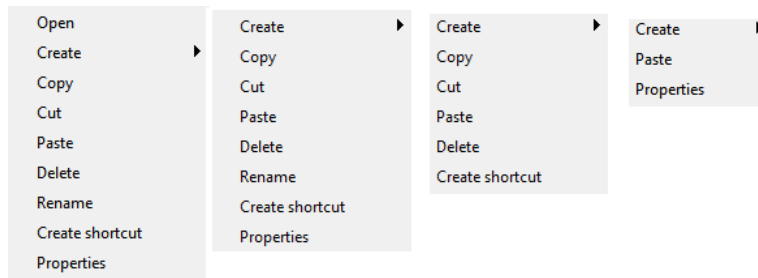


FIGURE 5.3: One file, one folder, multiselection and blank space context menus from left to right

Multiselection is also supported. This allows you to select multiple items and perform an operation for the entire group.

## 2. Button to open search filter

After clicking this button there appears search filter. To hide it you need to click this button once more. For detailed information about filtering, see ??

## 3. Button to open favorite paths window

After clicking this button there appears favorite paths window. It contains input field to add new path, button to confirm adding and container for existing favorites.

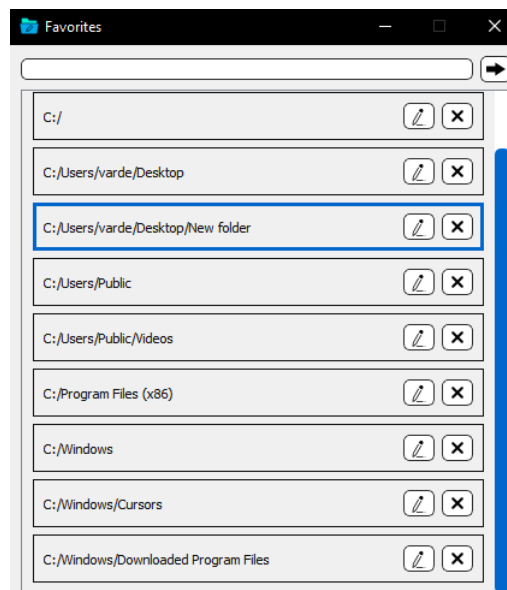


FIGURE 5.4: Favorite paths window

By default in input line current path is entered, which saves a lot of time for user as usually if someone wants to add folder to favorites, he is in it at the moment. After entering the path you need to press the button on the right or just Enter on the keyboard. If the path is valid, it will appear in the list, after which it can be interacted with in three ways. The first option is to go to the directory by left-clicking on the appropriate item. The second one is to delete favorite path by clicking on the cross button. The third one is to edit it, after which another validation will be performed. For detailed information about favorite paths feature, see [4.6](#)

#### 4. Navigation line

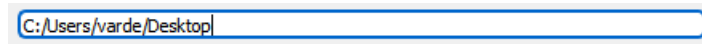


FIGURE 5.5: Navigation line

This line represents the path in the file system where the user is currently located. Thus, each time you go to the directory, the path written in this field will change. This will make it easier for the user to navigate. In addition, you can write some path in the navigation line and, if it is valid, there will be a transition to the appropriate directory. Each panel has its own line. For detailed information about navigation, see [4.4](#)

#### 5. Combobox for swapping drives



FIGURE 5.6: Combobox for swapping drives

After clicking, a popup list with all drives appears. Available on Windows only. For detailed information about navigation, see [4.4](#)

#### 6. "File" drop-down menu

Not used yet, added for options that will appear in the future.

#### 7. "Edit" drop-down menu

For now "Edit" drop-down menu contains option to call global search.

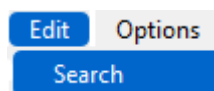


FIGURE 5.7: "Edit" drop-down menu

Search window contains three input fields for file name, containing text and target directory. After entering data user should press find button(or just Enter on the keyboard) and all the found files will be displayed as it is shown on the figure below. For detailed information about global search, see [4.5](#)

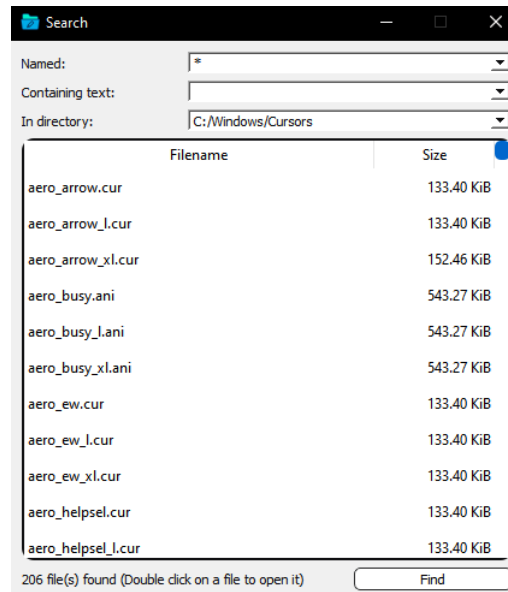


FIGURE 5.8: Global search window

## 8. "Options" drop-down menu

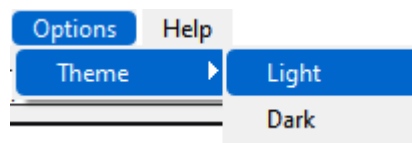


FIGURE 5.9: "Options" drop-down menu

For now, in "Options" menu one can change application theme.

## 9. "Help" drop-down menu

Not used yet, added for options that will appear in the future.

## 10. Cloud storage widget

This widget displays various information about cloud storage. After clicking on "Details" button, the distribution of file types on the drive is shown. To close this view "Close" button should be clicked. For detailed information about cloud drives support, see [4.7](#)

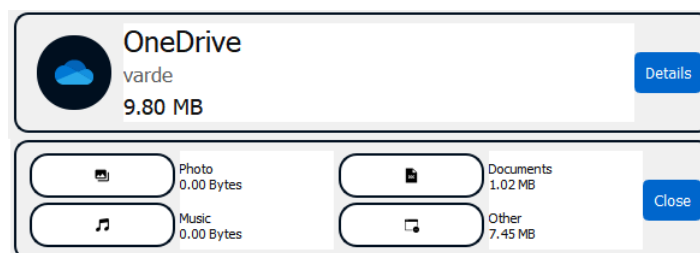


FIGURE 5.10: Two modes of cloud storage widget

## 11. Total memory chart



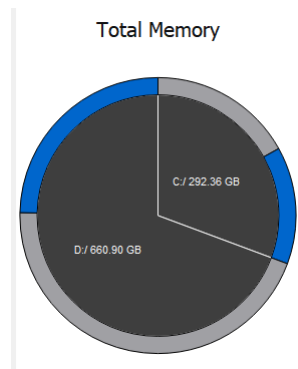


FIGURE 5.11: Total memory chart

The chart above shows distribution of total device memory. This is Pie Chart with few slices depending on amount of drives. For each of these slices there are two outer sections: used memory and free memory. So it can be easily determined how loaded each disk is. For detailed information about information panel, see [4.8](#)

## 12. Types distribution chart

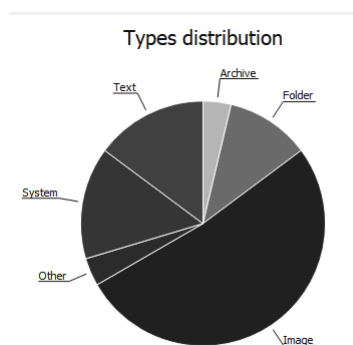


FIGURE 5.12: Types distribution chart

The chart shows the distribution of types in the folder. Updated when you move to another directory. Labels are shown outside the slices to make it easier to visually perceive a large number of types. For detailed information about information panel, see [4.8](#)

## 13. Drives information widget

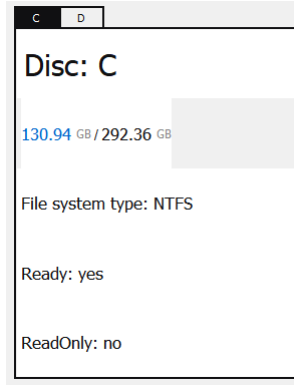
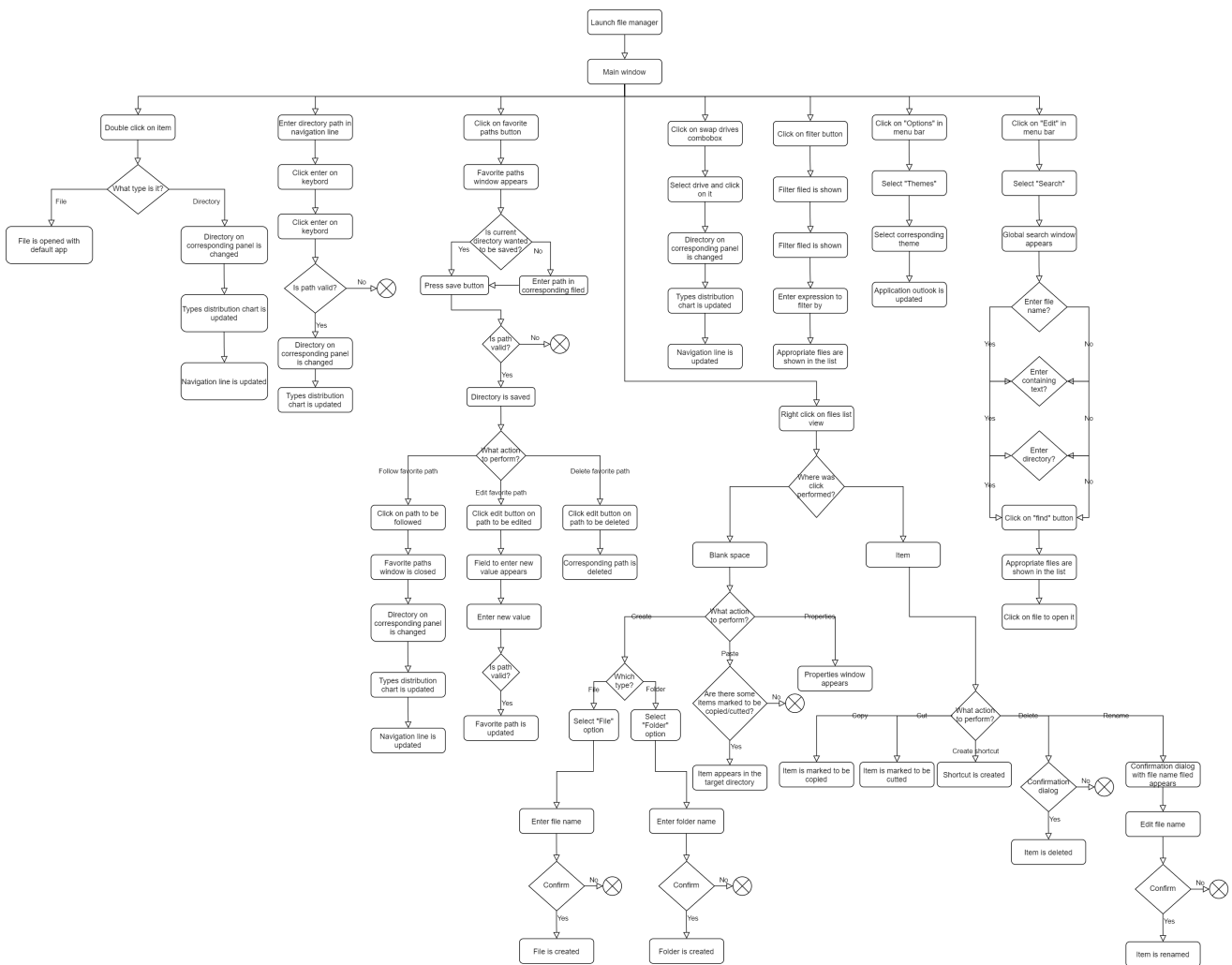


FIGURE 5.13: Drives information widget

Widget displays some information about all existing drives in the file system. User can swap between drives with buttons above. The active tab changes color to make it intuitive which one the user is currently on. For detailed information about information panel, see 4.8

## 5.2 User flow



## 5.3 Themes

Recently, the trend to add a dark theme to applications and sites has become popular. Many studies have already appeared on this topic, the opinion of scientists is as follows.

While the dark theme may provide a number of benefits for some visually impaired users - particularly those with cataracts, research results point to the benefit of positive polarity for users with normal vision. In other words, for users with normal vision, a light theme will lead to better performance in most cases.

These results are best explained by the fact that with positive contrast polarity there is more light and therefore more pupil contraction. The result is fewer spherical aberrations, greater depth of field, and an overall better ability to focus on detail without eye strain.

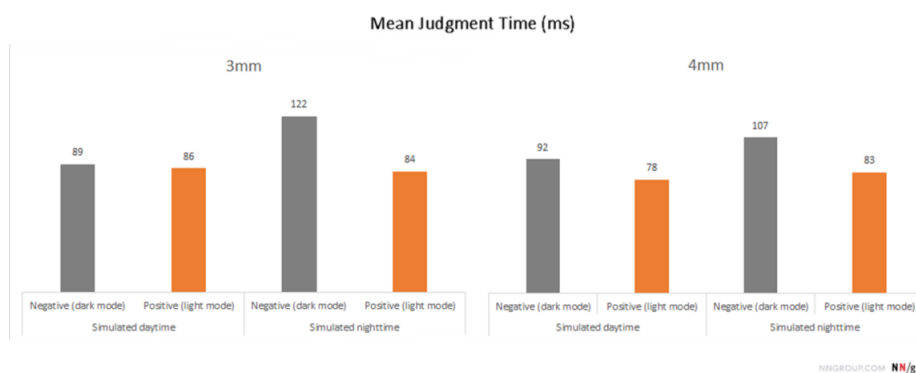


FIGURE 5.14

At the same time, it is necessary to allow users to switch to a dark theme at will - for three reasons: there may be long-term effects, with a light theme; some visually impaired people will work better on a dark theme; and some users just like the dark theme. (We know that people rarely change the default settings, but they should be able to do so.) It is unlikely that people will change the display mode for a random site, but if they are using a website or application, they may need this option. In particular, applications designed to be read in electronic forms (such as books, magazines, and even news sites) should suggest a feature theme to topics. And this option should ideally apply to all screens of this site or application. For more details, see *Dark Mode vs. Light Mode: Which Is Better?*

### 5.3.1 Light theme

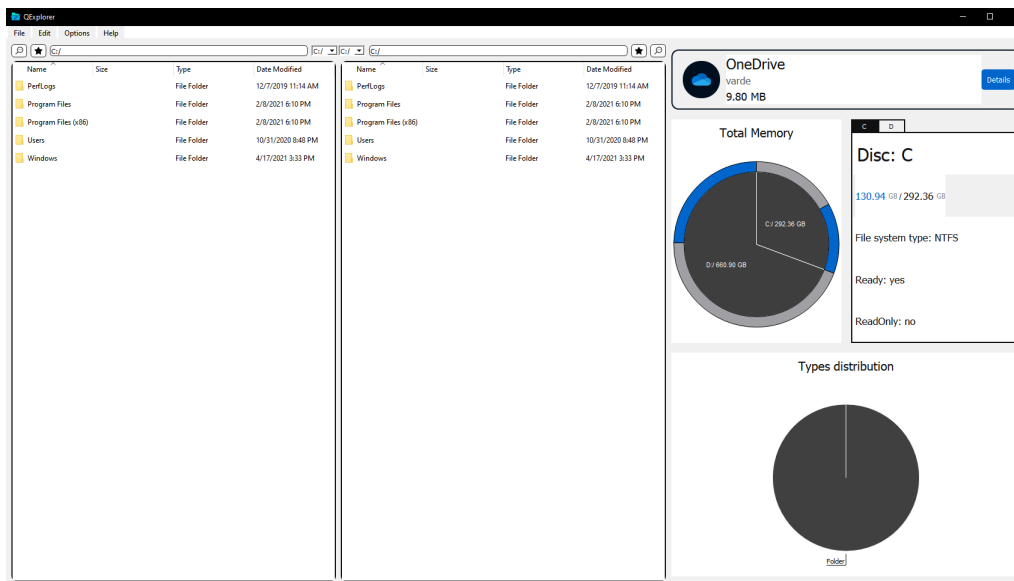


FIGURE 5.15: Light theme

The primary color used in this theme is  white(#FFFFFF). It covers about 85% of the application area. All the other elements such as symbols, borders, icons, etc. are mostly  black(#000000). When you activate some elements, they change their color to visually emphasize the user's action. This activation color is  royalblue(#0066CC). There are many borders in the interface. It is done to highlight elements and let user know exactly where he can interact with them and where he can not. It could be done using background color, as it is in dark theme case, but for the light one it was decided to leave this color the same for most elements, so the borders are the best choice.

### 5.3.2 Dark theme

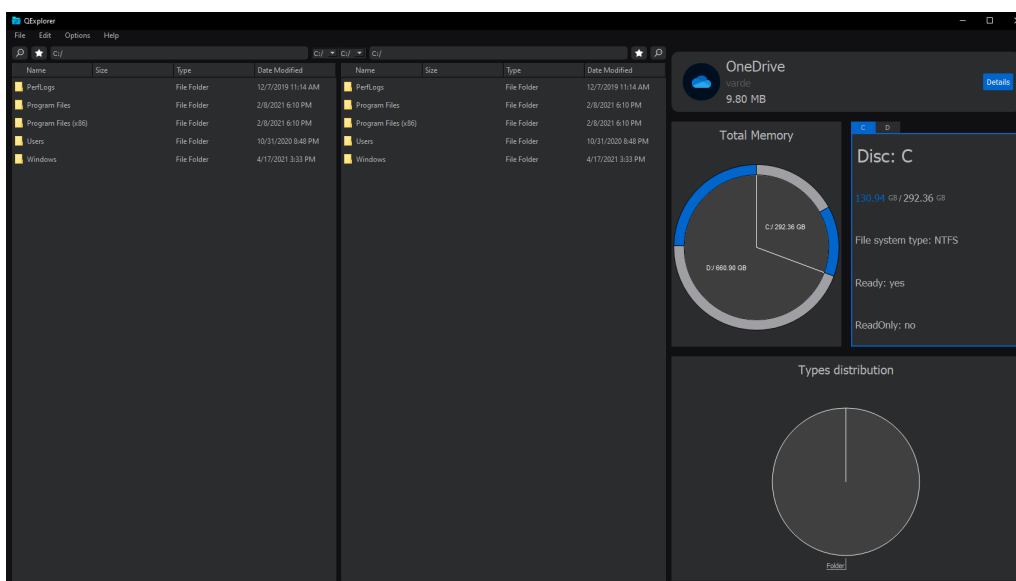






FIGURE 5.16: Dark theme

The primary color used in this theme is  darkslategray(#2B2C2D). It serves as a background color for all the main elements of the interface. In order to be able to distinguish the elements, the background color of the window was set to  black(#101012).  Silver(#BABABF) was chosen as the color of the font and icons. It was done to ensure that body text passes WCAG's AA standard of at least 4.5:1 when applied to surfaces at the highest (and lightest) elevation. In our case contrast ratio is equal to 7.23 which is pretty optimal. As with the light theme, there is an activation color that is also  royalblue(#0066CC).

## Chapter 6

# Evaluation

### 6.1 Testing environment

**OS:** Windows/Ubuntu

**CPU:** number of CPU cores: 8, number of threads: 16, base clock: 3.6GHz, L1 cache: 512KB, L2 cache: 4MB, L3 cache: 32MB

**Motherboard:** chipset: AMD X570, form factor: ATX, CPU support: AMD Socket AM4, memory support: dual-channel

**RAM:** memory type: DDR4, capacity: 16GB (8GBx2), tested speed: 3200MHz, tested latency: 16-18-18-38, SPD speed: 2133MHz

**SSD:** capacity: 1TB, read speed: up to: 550 MB, write speed: up to: 520 MB, cell type: TLC

### 6.2 Results

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	

FIGURE 6.1: Performance metrics

Class name	Lines	Methods	Class name	Lines	Methods
MainWindow	575	28	DeletionUtils	46	3
SearchWindow	179	11	CloudDriveUtils	42	3
StatisticsWidget	129	5	CloudDrive	41	8
Properties	98	17	FavoritePathsContainer	34	3
CreationUtils	89	5	MainSlice	30	5
ConfigParser	82	5	DiscView	22	2
StatisticsUtils	78	4	SwapDrivesUtils	22	2
CloudDriveWidget	73	6	CopyPathUtils	19	1
DonutBreakdownChart	73	4	FavoritesMainWindow	17	3
DirectorySizeCalculationUtils	66	5	NavigationUtils	15	1
CopyPasteUtils	56	3	FiltersUtils	15	1
RenameUtils	48	3	PropertiesWindow	12	2

TABLE 6.1: Code metrics #1

Type	Purpose	Number of lines
<b>.cpp</b>	Main business logic and application behavior definition	1956
<b>.h</b>	Declaration of different constructs, such as classes, enums, etc.	774
<b>.css</b>	Describing the presentation of an application	558
<b>.ui</b>	Definition of application components	811
<b>Total</b>		4099

TABLE 6.2: Code metrics #2

## Chapter 7

# Summary

### 7.1 Conclusion

In this work we have created fast, convenient and easy-to-use file manager, that meets all modern user needs. Quite a few things have not been implemented, for different reasons, but in general all expectations are justified. Some new features have been introduced in file manager development, that are not often seen in this area, such as info panel, favorites directories, few themes and so on. And still a lot of things that will be written below need to be improved and corrected.

### 7.2 Future work

#### Features to be implemented:

- **FTP Client**  
Program for easy access to FTP server. This way, the file manager becomes a FTP client through which you can receive files over the network. Feature has not yet been implemented, due to the fact that the new version of Qt has removed support for FTP, and switch to the previous version requires a lot of time to adapt the code.
- **Built-in command line**  
Command line interface(CLI) to interact with operating system by entering various commands. This feature was already in the project, but it had to be removed due to the large number of bugs and the amount of time required to fix them. The main problem is that the project is cross-platform, and API is significantly different on different operating systems.
- **Drag and drop**  
So far, only external drag&drop is supported. This means that we can drag and drop files into other programs, but we can't do it inside the file manager.

#### Known bugs:

- Names in columns is changing from bold to regular on click
- Cloud drives are not found on some pcs
- Parts of the charts are not visible on some screen resolutions
- The slider covers an important part of the interface
- Style issues: wrong borders, extra margins and paddings, violation of the color theme in some places, etc.



## Appendix A

# References

[qExplorer GitHub repository](#)

# Bibliography

- Cloud storage*. URL: [https://en.wikipedia.org/wiki/Cloud\\_storage](https://en.wikipedia.org/wiki/Cloud_storage).
- Dark Mode vs. Light Mode: Which Is Better?* URL: <https://www.nngroup.com/articles/dark-mode/>.
- Directory opus general features*. URL: <https://www.gpssoft.com.au/program/program.html>.
- File systems*. URL: [https://cms.ucu.edu.ua/pluginfile.php/135728/mod\\_resource/content/1/Lec\\_3\\_OS.pdf](https://cms.ucu.edu.ua/pluginfile.php/135728/mod_resource/content/1/Lec_3_OS.pdf).
- Nautilus general features*. URL: <https://help.ubuntu.ru/wiki/nautilus>.
- Official Qt documentation. QFileSystem*. URL: <https://doc.qt.io/qt-5/qfilesystemmodel.html>.
- Q-Dir general features*. URL: <http://www.softwareok.com/?seite=Freeware/Q-Dir>.
- Shortcuts*. URL: [https://en.wikipedia.org/wiki/Shortcut\\_\(computing\)](https://en.wikipedia.org/wiki/Shortcut_(computing)).
- Total commander general features*. URL: <https://www.ghisler.com/feature1.htm>.
- Windows Explorer general features*. URL: <https://www.informit.com/articles/article.aspx?p=2163343&seqNum=2>.