CSCE340102 - Operating systems

Operating System Project

Final Milestone

Name: Ali Yassine, Nermien Elassy, Anas Ibrahim

ID: 900204483, 900196006, 900204611

Supervised under: Dr. Amr El Kadi

Abstract:

This disk analyzer was developed from scratch using the Tauri framework. The backend relies solely on Rust and the GUI is a mixture of html, css and javascript. The disk analyzer supports user interaction where the user can interact with buttons to invoke the features and input directories.

Engineering Decisions:

It is well known that engineering decisions should also depend on history and past apps and experience. Due to this, a big part of what we did was done pre-developing the app itself. The main part was inspecting other disk analyzers such as WindirStat,JDiskReport, Filelight and other analyzers. We tested these analyzers using **black box texting** and **white box testing.** After black box testing, we were able to identify the following:

- The main features to implement
- GUI designs to implement
- GUI designs to stay away from

At this stage we were able to identify what features we wanted to implement and others that we wanted to stay away from due to several reasons including complications for the developers and the users.

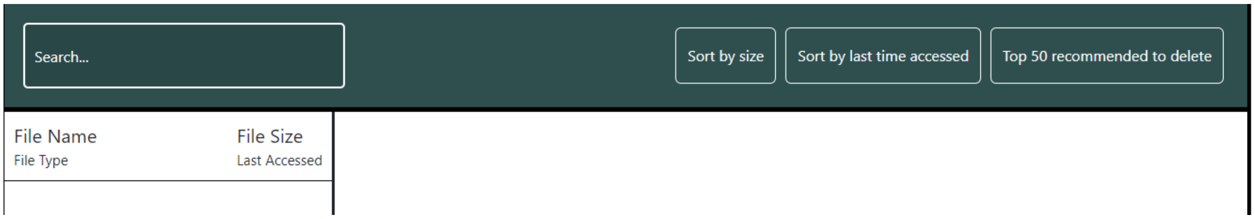
After **white box** testing and inspecting the hundreds of lines of codes of other apps and analyzers we were able to decide the following:

- What framework we want to use (Tauri)
- What languages we wanted for the frontend (html,css,javascript)
- What data structures to use (vectors, structs, vector of structs….)

- What crates to use (fs_extra,filesize,WalkDir…)

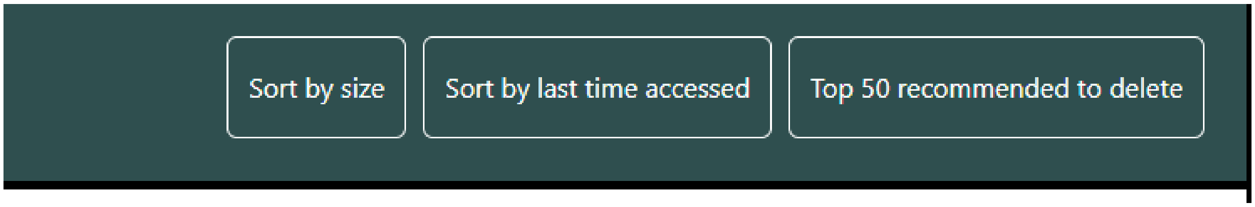
Features:

1-



- Users can enter the directory they want in the search box at the top of the page.
- The box under the search box is the template of how the files will be displayed.Thus, we will display the file's name, its size and when it was last accessed.

2- These 3 buttons have the below functionalities:



Sort by size: Prints the files sorted by size from greatest to smallest.

Sort by last time accessed: Prints the files sorted by last time accessed from oldest to latest.

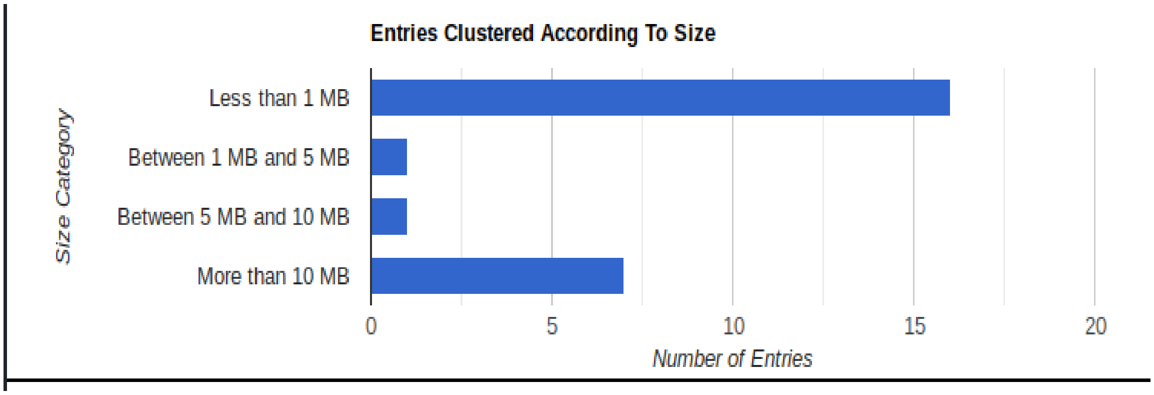
Top 50 recommended to delete: Recommends 50 files according to both size and when it was last accessed. Ex: If a file is big in size and also has not been accessed since a long time, we get the in the recommendations.

3- Statistics:

For the users to get a simple view about how the disk or a certain directory is distributed between files, we added a functionality that displays statistics using two bar charts.

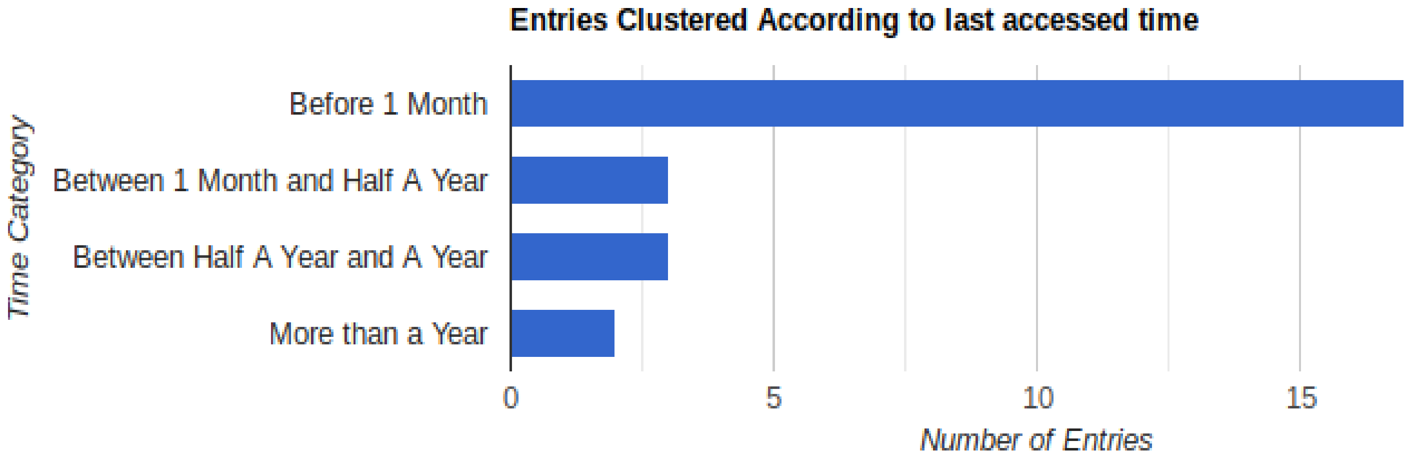
Bar Chart 1:

This bar chart displays the directory as clustered showing the number of files in each size category (less than 1Gb, less than 5Gb,less than 10Gb, others)



Bar Chart 2:

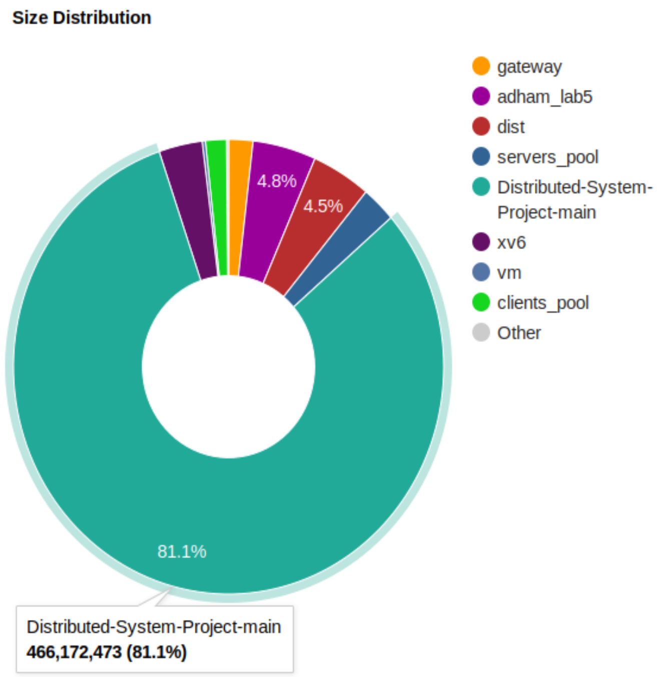
This bar chart displays the directory as clustered showing the number of files in each time category (less than 1 month ,less than 6 months ,less than a year , others)



Notes: Take into consideration that these groups do not interleave, thus there is no file that is double counted, thus a file that is less than 5Gb means that it is more than 1 Gb but less than 5Gb and so on.

4- Pie Chart

The user can also view how the files are distributed using a pie chart.



If walking through a directory called “Academics” that contains the below folders:

- OS
- Architecture
- Deep Learning
- Chemistry
- Biology
- Software Engineering

Then we get the pie chart above.

Tools:

This project was developed using the Tauri framework on visual studio code, where the backend was developed using Rust language and the frontend developed using html, css and javascript.

Algorithms:

- Using the “fs” crate (file system manipulation operations), we use the “read_dir” to walk through the directory.
- For each file or folder while looping we get the name, size and last time accessed and push them all into a **vector of structs** that holds info about each file or folder.
- In a new function we create a comparator and sort the vector of structs by size
- Another function creates a new comparator and sorts the vector of structs by last time accessed.
- The function top_10 walks through the vector of structs and compares according to both comparators, last time accessed and size , and recommends ten files to delete.

Challenges:

- Rust language is a young language and it is still developing, thus the documentation was not a lot which made the research and debugging part harder than other languages such as c++.
- We tried to use FLTK for the GUI but it was a bit complicated and not well known between developers and since it is part of Rust, the documentation was also limited.

- Building a Tree in the backend was doable and we started by building a tree but to develop a fast application we decided to drop the tree and use other data structures where recursive traversal is limited.
- Linking between the backend and the GUI was a challenge in itself because none of the developers in the team had used javascript before.

Roles:

Ali Yassine: Backend

Anas Ibrahim: Linking between GUI and Backend

Nermien El Agamy: GUI