# REPORT:

## 📁 File Explorer – Capstone Project  Project Overview

This project is a simple **File Explorer Application** built using **C++ (C++17)** .It enables users to explore, list, and manage files and directories through a command line interface.

The project showcases the uses of the C++ filesystem library for performing dynamic file and directory operations .It serves as an excellent learning example learning example for understanding how real-world file management systems function.

---

## Objective

To design and implement a console-based **File Explorer** that allows:

- Navigating through directory structures.
- Displaying detailed file and folder information.
- Performing file operations like creating, renaming, and deleting files/folders.
- Showcasing practical implementation of the <filesystem>library in C++.

---

## Technologies Used

- C++17
- STL (Standard Template Library)
- Library for file and directory handling
- Command**-Line Interface (CLI)**

---

## How to Run

1. Open **Git Bash** or **Command Prompt** inside your project folder.
2. Compile the code using: g++ file_explorer.cpp -o file_explorer -lstdc++fs

# NOVALITY:

**Novelty (Highlights)**

- Introduced **advanced command-line operations** such as:

- ☑ **Regex-based search** — allows users to search files dynamically using keyword patterns across all subdirectories.

- ☑ **Permission control (chmod)** — enables modifying file access rights similar to real Unix/Linux systems.

- Enhanced usability by providing **detailed file metadata** including type, size, and permissions.

- Supports **recursive directory traversal**, enabling deeper file exploration beyond basic listing.

- Designed as an **educational mini-shell system**, integrating real OS-level functionalities using modern C++17 <filesystem> capabilities.

# CODE:

```
#include <iostream>

#include <string>

#include <vector>

#include <filesystem>

#include <fstream>

#include <deque>

#include <regex>

#include <sstream>

#include <system_error>

#include <iomanip>


namespace fs = std::filesystem;


struct FileInfo {


   std::string name;

std::string path;     bool
```

```
is_dir = false;
uintmax_t size = 0;
std::string perms;
};

std::string perms_to_string(fs::perms p) {    //
Show rwx for owner/group/others (9 chars)
   std::string s = "---------";
   s[0] = (p & fs::perms::owner_read)  != fs::perms::none ? 'r' : '-';
s[1] = (p & fs::perms::owner_write) != fs::perms::none ? 'w' : '-';
s[2] = (p & fs::perms::owner_exec)  != fs::perms::none ? 'x' : '-';
s[3] = (p & fs::perms::group_read)  != fs::perms::none ? 'r' : '-';    s[4]
= (p & fs::perms::group_write) != fs::perms::none ? 'w' : '-';    s[5] =
(p & fs::perms::group_exec)  != fs::perms::none ? 'x' : '-';    s[6] = (p
& fs::perms::others_read) != fs::perms::none ? 'r' : '-';    s[7] = (p &
fs::perms::others_write)!= fs::perms::none ? 'w' : '-';    s[8] = (p &
fs::perms::others_exec) != fs::perms::none ? 'x' : '-';    return s;
}

std::vector<FileInfo> list_directory(const std::string &path) {
std::vector<FileInfo> out;    std::error_code ec;
fs::directory_iterator it(path, ec);
   if (ec) {
      std::cerr << "ls: cannot access '" << path << "': " << ec.message() << '\n';
      return out;
   }
   for (const auto &entry : it) {
FileInfo fi;
```

```
        fi.name = entry.path().filename().string();
    fi.path = entry.path().string();
    std::error_code ec2;        auto st =
    entry.symlink_status(ec2);
        if (!ec2) {            fi.is_dir =
    fs::is_directory(st);
            try {
                if (fs::is_regular_file(st)) {
    fi.size = fs::file_size(entry.path(), ec2);
                    if (ec2) fi.size = 0;
                } else {
       fi.size = 0;
                }
                fi.perms = perms_to_string(st.permissions());
            } catch (...) {
       fi.size = 0;
                fi.perms = "?????????";
            }
        } else {
    fi.is_dir = false;
    fi.size = 0;
            fi.perms = "?????????";
        }
        out.push_back(fi);
      }
      return out;
    }
```

```
void print_listing(const std::vector<FileInfo>& entries) {
std::cout << std::left << std::setw(30) << "Name"
          << std::setw(8) << "Type"
          << std::setw(12) << "Size"          <<
"Perms\n";    for (auto &e : entries) {       std::cout
<< std::left << std::setw(30) << e.name
             << std::setw(8) << (e.is_dir ? "DIR" : "FILE")
             << std::setw(12) << e.size
             << e.perms << "\n";
   }
}

bool change_directory(const std::string &path) { std::error_code ec;   fs::current_path(path,
ec);
   if (ec) {
      std::cerr << "cd: " << ec.message() << '\n';
     return false;
   }
   return true;
}

std::string get_current_path() {
std::error_code ec;    auto p =
fs::current_path(ec);    if (ec)
return std::string{};    return
p.string();
}
```

```
bool create_file(const std::string &path) {
std::ofstream ofs(path, std::ios::out | std::ios::app);
   if (!ofs) {
      std::cerr << "mkfile: failed to create " << path << '\n';
return false;
   }
   return true;
}

bool create_directory(const std::string &path) {
   std::error_code ec; bool ok =
   fs::create_directories(path, ec);
   if (ec) {
      std::cerr << "mkdir: " << ec.message() << '\n';
    return false;
   }
   return ok || fs::exists(path);
}

bool remove_path(const std::string &path) {
std::error_code ec;        uintmax_t removed =
fs::remove_all(path, ec);
   if (ec) {
      std::cerr << "rm: " << ec.message() << '\n';
      return false;
   }
   return removed > 0;
}
```

```
bool copy_path(const std::string &src, const std::string &dst) {     std::error_code ec;
fs::copy(src, dst, fs::copy_options::recursive | fs::copy_options::overwrite_existing, ec);
   if (ec) {
      std::cerr << "cp: " << ec.message() << '\n';
      return false;
   }
   return true;
}

bool move_path(const std::string &src, const std::string &dst) {
std::error_code ec;      fs::rename(src, dst, ec);
   if (!ec) return true;

  // If rename fails (different FS or permission), try copy+remove
if (!copy_path(src, dst)) return false;    return remove_path(src);
}

bool show_file_head(const std::string &path, int lines) {
   std::ifstream ifs(path);
   if (!ifs) {
      std::cerr << "cathead: cannot open " << path << '\n';
      return false;
   }
   std::string line;     int count = 0;     while
(count < lines && std::getline(ifs, line)) {
std::cout << line << '\n'; ++count;
   }
   return true;
}
```

```
bool show_file_tail(const std::string &path, int lines) {
std::ifstream ifs(path);
   if (!ifs) {
      std::cerr << "cattail: cannot open " << path << '\n';
      return false;
   }
   std::deque<std::string> buffer;
   std::string line;
   while (std::getline(ifs, line)) {
   buffer.push_back(line);     if ((int)buffer.size() >
   lines) buffer.pop_front();
   }
   for (auto &l : buffer) std::cout << l << '\n';
return true;
}

std::vector<std::string> search_recursive(const std::string &root, const std::string &pattern) {
std::vector<std::string> results;     std::regex re;
   try {
      re = std::regex(pattern, std::regex::ECMAScript | std::regex::icase);
   } catch (const std::exception &e) {        std::cerr <<
"search: invalid regex: " << e.what() << '\n';        return
results;
   }
   std::error_code ec;
   for (auto it = fs::recursive_directory_iterator(root, ec); !ec && it !=
fs::recursive_directory_iterator(); it.increment(ec)) {
      if (ec) break;
```

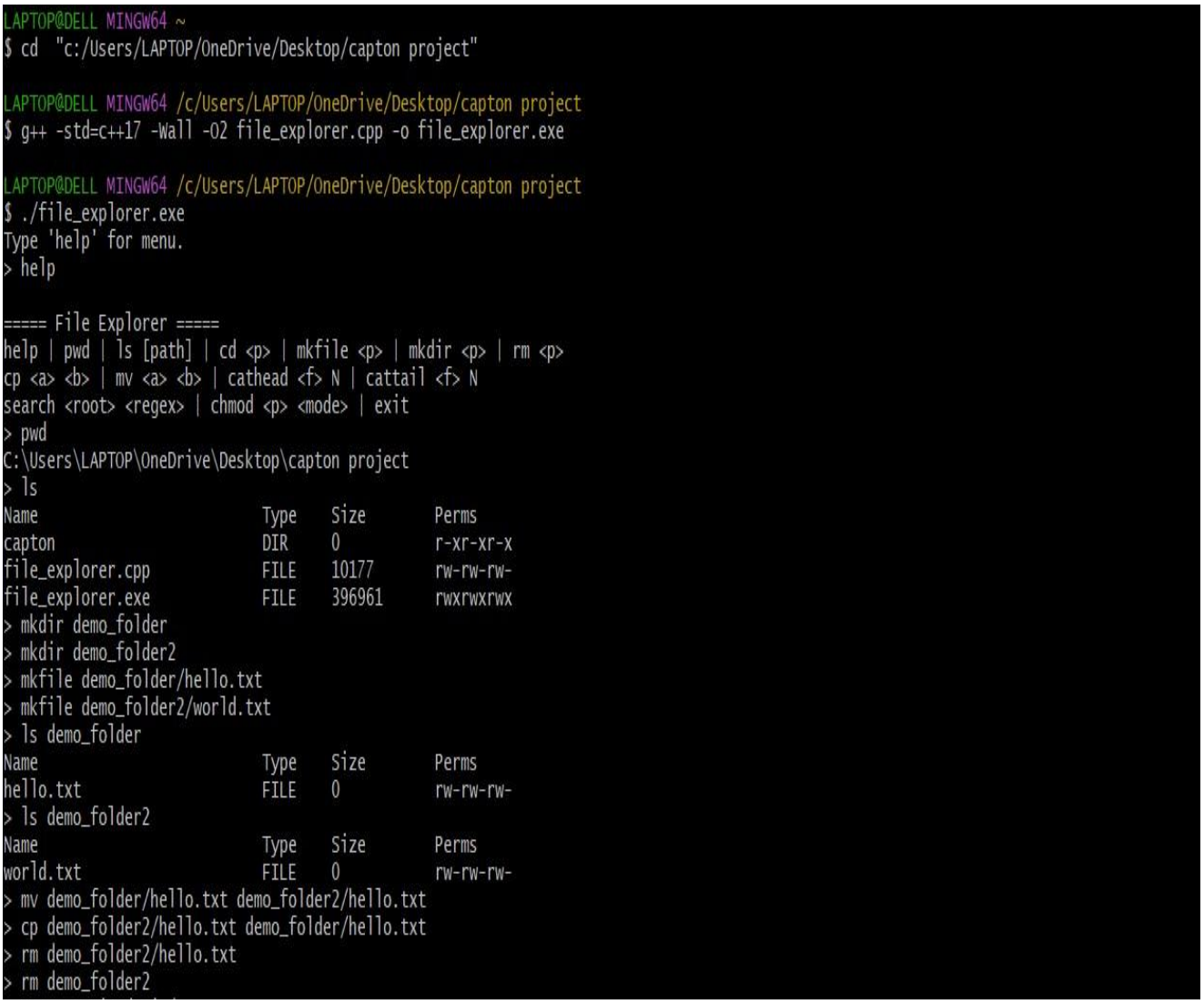
```
        try {
            if (std::regex_search(it->path().filename().string(), re)) {                results.push_back(it-
>path().string());
            }
        } catch (...) {}
    }
    if (ec) std::cerr << "search: " << ec.message() << '\n';
return results;
}
// Convert string like "755" (octal) to perms and apply (owner/group/others rwx).
bool change_permissions(const std::string &path, const std::string &mode) {
    if (mode.size() != 3) {        std::cerr << "chmod: mode
should be 3 digits like 755\n";
        return false;
    }
    int m = 0;
    try {
        m = std::stoi(mode, nullptr, 8); // parse as octal
    } catch (...) {
        std::cerr << "chmod: invalid mode\n";
        return false;
    }

    fs::perms p = fs::perms::none;
    // owner     if (m & 0400) p |=
fs::perms::owner_read;     if (m & 0200) p |=
fs::perms::owner_write;     if (m & 0100) p |=
fs::perms::owner_exec;
```

```
   // group     if (m & 0040) p |=
fs::perms::group_read;     if (m & 0020) p |=
fs::perms::group_write;     if (m & 0010) p |=
fs::perms::group_exec;
  // others    if (m & 0004) p |=
fs::perms::others_read; if (m & 0002) p |=
fs::perms::others_write; if (m & 0001) p |=
fs::perms::others_exec; std::error_code ec;
fs::permissions(path, p, ec);
   if (ec) {
      std::cerr << "chmod: " << ec.message() << '\n';
      return false;
   }
   return true;
}

void show_menu() {     std::cout << "\n===== File Explorer =====\n";     std::cout <<
"help | pwd | ls [path] | cd <p> | mkfile <p> | mkdir <p> | rm <p>\n";     std::cout <<
"cp <a> <b> | mv <a> <b> | cathead <f> N | cattail <f> N\n";     std::cout << "search
<root> <regex> | chmod <p> <mode> | exit\n";
}

int main() {
std::string line;
   std::cout << "Type 'help' for menu.\n";
   while (true) {        std::cout << "> ";
if (!std::getline(std::cin, line)) break;
if (line.empty()) continue;
```

```
    std::istringstream iss(line);
std::string cmd;       iss >>
cmd;


     if (cmd == "help") { show_menu(); }
      else if (cmd == "pwd") { std::cout << get_current_path() << '\n'; }
   else if (cmd == "ls") {        std::string p;
         if (iss >> std::ws && std::getline(iss, p) && !p.empty()) {
            // trim
            if (p.front() == ' ') p.erase(0, p.find_first_not_of(' '));
print_listing(list_directory(p));
         } else {
            print_listing(list_directory(get_current_path()));
         }
      }
      else if (cmd == "cd") {           std::string p; if (iss
>> p) change_directory(p);
      }
      else if (cmd == "mkfile") {
std::string p; if (iss >> p) create_file(p);
      }
      else if (cmd == "mkdir") {           std::string p; if
(iss >> p) create_directory(p);
      }
      else if (cmd == "rm") {           std::string p; if
(iss >> p) remove_path(p);
      }
      else if (cmd == "cp") {          std::string a, b; if (iss
 >> a >> b) copy_path(a, b);
```

```
        }
        else if (cmd == "mv") {           std::string a, b; if (iss
>> a >> b) move_path(a, b);
        }
        else if (cmd == "cathead") {        std::string f; int n; if (iss
    >> f >> n) show_file_head(f, n);
        }
        else if (cmd == "cattail") {           std::string f; int n; if
(iss >> f >> n) show_file_tail(f, n);
        }
        else if (cmd == "search") {
            std::string r, p;
if (iss >> r >> p) {
                for (auto &x : search_recursive(r, p)) std::cout << x << '\n';
            } else {
                std::cerr << "search: usage: search <root> <regex>\n";
            }
        }
        else if (cmd == "chmod") {           std::string p, m; if (iss >> p
>> m) change_permissions(p, m);
        }
        else if (cmd == "exit") break;
        else {
            std::cerr << "Unknown command: " << cmd << " (type 'help')\n";
        }
    }
    return 0;
}
```

SCREENSHOT:



```
LAPTOP@DELL MINGW64 ~
$ cd  "c:/Users/LAPTOP/OneDrive/Desktop/capton project"

LAPTOP@DELL MINGW64 /c/Users/LAPTOP/OneDrive/Desktop/capton project
$ g++ -std=c++17 -Wall -O2 file_explorer.cpp -o file_explorer.exe

LAPTOP@DELL MINGW64 /c/Users/LAPTOP/OneDrive/Desktop/capton project
$ ./file_explorer.exe
Type 'help' for menu.
> help

===== File Explorer =====
help | pwd | ls [path] | cd <p> | mkfile <p> | mkdir <p> | rm <p>
cp <a> <b> | mv <a> <b> | cathead <f> N | cattail <f> N
search <root> <regex> | chmod <p> <mode> | exit
> pwd
C:\Users\LAPTOP\OneDrive\Desktop\capton project
> ls
Name                          Type    Size        Perms
capton                        DIR     0           r-xr-xr-x
file_explorer.cpp             FILE    10177       rw-rw-rw-
file_explorer.exe             FILE    396961      rwxrwxrwx
> mkdir demo_folder
> mkdir demo_folder2
> mkfile demo_folder/hello.txt
> mkfile demo_folder2/world.txt
> ls demo_folder
Name                          Type    Size        Perms
hello.txt                     FILE    0           rw-rw-rw-
> ls demo_folder2
Name                          Type    Size        Perms
world.txt                     FILE    0           rw-rw-rw-
> mv demo_folder/hello.txt demo_folder2/hello.txt
> cp demo_folder2/hello.txt demo_folder/hello.txt
> rm demo_folder2/hello.txt
> rm demo_folder2
```