

CAPSTONE PROJECT REPORT

FILE EXPLORER APPLICATION

(Console-Based Linux OS)

Assignment 1 - Linux OS

Submitted by:

Name: Rakesh Kumar Raut

Regd No: 2241018094

Department: CSIT

Course: B. Tech

Institution: Institute of Technical Education and Research(ITER)

Date of Submission: 9/11/25

1. PROJECT OVERVIEW

1.1 OBJECTIVE

The primary objective of this project is to develop a console-based File Explorer Application in C++ that interfaces with the Linux operating system to efficiently manage files and directories.

This application demonstrates:

- Operating System integration with Linux file system
- System programming using C++ and POSIX APIs
- File and directory manipulation operations
- User interaction through console-based interface

1.2 PROJECT DESCRIPTION

The File Explorer Application is a command-line tool that provides users with comprehensive file management capabilities. It allows users to navigate through directories, perform file operations (create, copy, delete), search for files, and view detailed file information including permissions.

Key Features:

- Interactive menu-driven interface

- Real-time directory navigation
- File manipulation (create, copy, delete)
- Advanced file search functionality
- File permission viewing
- Error handling and validation

1.3 TECHNOLOGY STACK

Programming Language: C++ (Standard C++11)

Operating System: Linux (Ubuntu/Debian-based)

Development Tools: g++ compiler

Libraries Used:

- `iostream` - Input/output operations
 - `dirent.h` - Directory operations
 - `sys/stat.h` - File statistics
 - `fstream` - File stream operations
 - `string` - String manipulation
 - `cstdio` - C standard input/output
 - `unistd.h` - POSIX API access
 - `time.h` - Time functions
-
-

2. FEATURES IMPLEMENTED

2.1 DISPLAY FILES (Option 1)

Description:

Lists all files and folders in the current directory with detailed information including name, type, and size.

Technical Implementation:

- Uses `opendir()` and `readdir()` from `dirent.h`
- Filters hidden files (starting with `'.'`)
- Uses `stat()` to retrieve file metadata
- Distinguishes files from directories using `d_type`

Key Code:

```
DIR* directory = opendir(current_directory.c_str());
struct dirent* file_entry;
while ((file_entry = readdir(directory)) != NULL) {

}
```

Features:

- ✓ Displays file names
- ✓ Shows file type (FILE/FOLDER)

- ✓ Shows file size in bytes
- ✓ Hides system files (starting with .)
- ✓ Formatted table output

2.2 DIRECTORY NAVIGATION (Options 2 & 3)

2.2.1 Open Folder (Option 2)

Description:

Allows users to navigate into subdirectories by entering folder name.

Technical Implementation:

- Validates folder existence before navigation
- Updates current_directory path
- Error handling for non-existent folders

Key Code:

```
string new_path = current_directory + "/" + folder_name;
DIR* test_dir = opendir(new_path.c_str());
if (test_dir != NULL) {
    current_directory = new_path;
}
```

Features:

- ✓ Path validation
- ✓ Real-time directory update
- ✓ Error messages for invalid paths

2.2.2 Go Back (Option 3)

Description:

Navigate to parent directory (one level up).

Technical Implementation:

- Parses current path to find parent
- Uses string manipulation to extract parent path
- Handles root directory edge case

Key Code:

```
int last_slash = current_directory.find_last_of('/');
if (last_slash > 0) {
    current_directory = current_directory.substr(0, last_slash);
}
```

Features:

- ✓ Safe backward navigation
- ✓ Root directory protection

- ✓ Path display after navigation

2.3 FILE CREATION (Option 4)

Description:

Creates new text files with user-provided content.

Technical Implementation:

- Uses ofstream for file creation
- Multi-line content input support
- END keyword to terminate input
- File handle validation

Key Code:

```
ofstream new_file(full_path.c_str());
string line;
while (getline(cin, line) && line != "END") {
    new_file << line << "\n";
}
new_file.close();
```

Features:

- ✓ Interactive file creation
- ✓ Multi-line content support
- ✓ EOF marker (END keyword)
- ✓ Success/failure feedback

2.4 FILE DELETION (Option 5)

Description:

Permanently deletes specified files from the file system.

Technical Implementation:

- Uses remove() function from cstdio
- Validates deletion operation
- Error handling for missing files

Key Code:

```
string full_path = current_directory + "/" + file_name;
int result = remove(full_path.c_str());
if (result == 0) {
    cout << "File deleted successfully!\n";
}
```

Features:

- ✓ File deletion capability

- ✓ Error handling
- ✓ Confirmation messages

2.5 FILE COPY (Option 6)

Description:

Creates a duplicate copy of a file with a new name.

Technical Implementation:

- Binary file reading for universal compatibility
- Character-by-character copy using get() and put()
- Source file validation
- Destination file creation

Key Code:

```
ifstream source(source_path, ios::binary);
ofstream dest(dest_path, ios::binary);
char ch;
while (source.get(ch)) {
    dest.put(ch);
}
```

Features:

- ✓ Text file copying
- ✓ Binary file support
- ✓ Source validation
- ✓ Content integrity preservation

2.6 FILE SEARCH (Option 7)

Description:

Searches for files and folders matching a search term.

Technical Implementation:

- Uses string::find() for pattern matching
- Iterates through directory entries
- Counts matching results
- Identifies file types in results

Key Code:

```
if (name.find(search_word) != string::npos) {
    cout << "Found: " << name;
    if (entry->d_type == DT_DIR) {
```

```
        cout << " [FOLDER]";
    }
    found_count++;
}
```

Features:

- ✓ Pattern-based search
- ✓ Case-sensitive matching
- ✓ Result counter
- ✓ File/folder type display

2.7 FILE DETAILS & PERMISSIONS (Option 8)

Description:

Displays comprehensive file information including size, permissions, and modification time.

Technical Implementation:

- Uses `stat()` to retrieve file metadata
- Extracts permission bits using bitwise operations
- Converts timestamp to readable format
- Error handling for missing files

Key Code:

```
struct stat file_stats;
stat(full_path.c_str(), &file_stats);

if (file_stats.st_mode & S_IRUSR) cout << "r";
if (file_stats.st_mode & S_IWUSR) cout << "w";
if (file_stats.st_mode & S_IXUSR) cout << "x";
```

Features:

- ✓ File size display
- ✓ Read/Write/Execute permissions
- ✓ Last modification timestamp
- ✓ File metadata access

2.8 INPUT VALIDATION & ERROR HANDLING

Description:

Robust error handling to prevent crashes and infinite loops.

Technical Implementation:

- cin.fail() checking for invalid input
- cin.clear() to reset error flags
- cin.ignore() to flush input buffer
- Continue loop on error

Key Code:

```
cin >> user_choice;
if (cin.fail()) {
    cin.clear();
    cin.ignore(10000, '\n');
    cout << "ERROR: Please enter valid number!\n";
    continue;
}
```

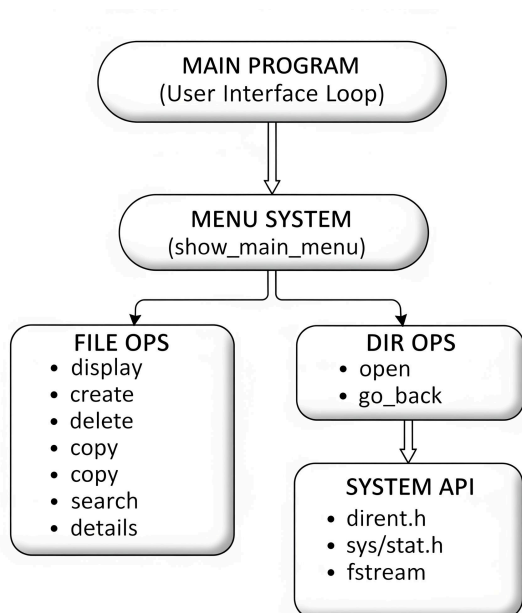
Features:

- ✓ Invalid input handling
- ✓ No infinite loops
- ✓ User-friendly error messages
- ✓ Program stability

3. SYSTEM ARCHITECTURE

3.1 Architecture Overview

The application follows a modular, function-based architecture:



3.2 Key Functions

Function Name

Purpose

<code>main()</code>	Program entry point, main loop
<code>show_main_menu()</code>	Display menu options
<code>display_files()</code>	List files and directories
<code>open_folder()</code>	Navigate into subdirectory
<code>go_back_folder()</code>	Navigate to parent directory
<code>make_new_file()</code>	Create new file with content
<code>remove_file()</code>	Delete specified file
<code>copy_a_file()</code>	Duplicate file
<code>search_file()</code>	Search for files by name
<code>show_file_details(ls())</code>	Display file metadata and permissions

Source Code:

```
#include <iostream>
#include <dirent.h>
#include <sys/stat.h>
```

```

#include <string>
#include <fstream>
#include <cstdio>
#include <unistd.h>
#include <time.h>

using namespace std;

string current_directory = ".";

void display_files()
{
    DIR* directory;
    directory = opendir(current_directory.c_str());

    if (directory == NULL) {
        cout << "ERROR: Could not open directory!" << endl;
        return;
    }

    cout << "\n--- Files in " << current_directory << " ---\n";
    cout << "Name\t\t\tType\t\tSize\n";
    cout << "-----\n";

    struct dirent* file_entry;
    file_entry = readdir(directory);

    while (file_entry != NULL) {
        string file_name = file_entry->d_name;

        if (file_name[0] != '.') {
            string complete_path = current_directory + "/" + file_name;
            struct stat file_info;
            stat(complete_path.c_str(), &file_info);

            cout << file_name << "\t\t";

            if (file_entry->d_type == DT_DIR) {
                cout << "[FOLDER]\t";
            } else {
                cout << "[FILE]\t\t";
            }

            cout << file_info.st_size << " bytes\n";
        }

        file_entry = readdir(directory);
    }
}

```

```

    }

    closedir(directory);
    cout << "\n";
}

void open_folder()
{
    string folder_name;
    cout << "Enter folder name: ";
    getline(cin, folder_name);

    string new_path = current_directory + "/" + folder_name;

    DIR* test_dir;
    test_dir = opendir(new_path.c_str());

    if (test_dir == NULL) {
        cout << "ERROR: Folder does not exist!\n";
        return;
    }

    closedir(test_dir);
    current_directory = new_path;
    cout << "Moved to: " << current_directory << "\n";
}

void go_back_folder()
{
    int last_slash = -1;
    for (int i = 0; i < current_directory.length(); i++) {
        if (current_directory[i] == '/') {
            last_slash = i;
        }
    }

    if (last_slash > 0) {
        current_directory = current_directory.substr(0, last_slash);
    } else {
        current_directory = ".";
    }

    cout << "Moved back to: " << current_directory << "\n";
}

void make_new_file()
{

```

```

    string file_name;
    cout << "Enter new file name: ";
    getline(cin, file_name);

    string full_path = current_directory + "/" + file_name;

    ofstream new_file;
    new_file.open(full_path.c_str());

    if (!new_file.is_open()) {
        cout << "ERROR: Cannot create file!\n";
        return;
    }

    cout << "File created successfully!\n";
    cout << "Enter file content (type 'END' on a new line to
finish):\n";

    string line;
    while (true) {
        getline(cin, line);
        if (line == "END") {
            break;
        }
        new_file << line << "\n";
    }

    new_file.close();
    cout << "File saved successfully!\n";
}

void remove_file()
{
    string file_name;
    cout << "Enter file name to delete: ";
    getline(cin, file_name);

    string full_path = current_directory + "/" + file_name;

    int result = remove(full_path.c_str());

    if (result == 0) {
        cout << "File deleted successfully!\n";
    } else {
        cout << "ERROR: Could not delete file!\n";
    }
}

```

```

void copy_a_file()
{
    string source_name, destination_name;

    cout << "Enter source file name: ";
    getline(cin, source_name);

    cout << "Enter destination file name: ";
    getline(cin, destination_name);

    string source_path = current_directory + "/" + source_name;
    string dest_path = current_directory + "/" + destination_name;

    ifstream source_file;
    source_file.open(source_path.c_str(), ios::binary);

    if (!source_file.is_open()) {
        cout << "ERROR: Source file not found!\n";
        return;
    }

    ofstream dest_file;
    dest_file.open(dest_path.c_str(), ios::binary);

    if (!dest_file.is_open()) {
        cout << "ERROR: Cannot create destination file!\n";
        source_file.close();
        return;
    }

    char ch;
    while (source_file.get(ch)) {
        dest_file.put(ch);
    }

    source_file.close();
    dest_file.close();

    cout << "File copied successfully!\n";
}

void search_file()
{
    string search_word;
    cout << "Enter search term: ";
    getline(cin, search_word);
}

```

```

    cout << "\nSearching for files containing: " << search_word <<
"\n";
    cout << "-----\n";

    DIR* dir = opendir(current_directory.c_str());
    if (dir == NULL) {
        cout << "ERROR: Cannot search!\n";
        return;
    }

    struct dirent* entry;
    int found_count = 0;

    entry = readdir(dir);
    while (entry != NULL) {
        string name = entry->d_name;

        if (name[0] != '.') {
            if (name.find(search_word) != string::npos) {
                cout << "Found: " << name;
                if (entry->d_type == DT_DIR) {
                    cout << " [FOLDER]";
                }
                cout << "\n";
                found_count++;
            }
        }

        entry = readdir(dir);
    }

    closedir(dir);

    if (found_count == 0) {
        cout << "No files found!\n";
    } else {
        cout << "\nTotal found: " << found_count << " file(s)\n";
    }
}

void show_file_details()
{
    string file_name;
    cout << "Enter file name: ";
    getline(cin, file_name);
}

```

```

string full_path = current_directory + "/" + file_name;

struct stat file_stats;
int result = stat(full_path.c_str(), &file_stats);

if (result != 0) {
    cout << "ERROR: File not found!\n";
    return;
}

cout << "\n--- File Information ---\n";
cout << "File Name: " << file_name << "\n";
cout << "File Size: " << file_stats.st_size << " bytes\n";

cout << "Permissions: ";
if (file_stats.st_mode & S_IRUSR) cout << "r"; else cout << "-";
if (file_stats.st_mode & S_IWUSR) cout << "w"; else cout << "-";
if (file_stats.st_mode & S_IXUSR) cout << "x"; else cout << "-";
cout << "\n";

cout << "Last Modified: " << ctime(&file_stats.st_mtime);
cout << "\n";
}

void show_main_menu()
{
    cout << "\n";
    cout << "=====\n";
    cout << "          FILE EXPLORER PROGRAM\n";
    cout << "=====\n";
    cout << "Current Location: " << current_directory << "\n";
    cout << "=====\n";
    cout << "1. Show all files\n";
    cout << "2. Open a folder\n";
    cout << "3. Go back to previous folder\n";
    cout << "4. Create new file\n";
    cout << "5. Delete a file\n";
    cout << "6. Copy a file\n";
    cout << "7. Search for files\n";
    cout << "8. Show file details\n";
    cout << "9. Exit program\n";
    cout << "=====\n";
    cout << "Enter your choice: ";
}

int main()
{
    cout << "\n*** Welcome to File Explorer! ***\n";

```

```
cout << "This program helps you manage files on your computer.\n";

int user_choice;

while (true) {
    show_main_menu();
    cin >> user_choice;

    if (cin.fail()) {
        cin.clear();
        cin.ignore(10000, '\n');
        cout << "ERROR: Please enter a valid number (1-9)!\n";
        continue;
    }

    cin.ignore();
    if (user_choice == 1) {
        display_files();
    }
    else if (user_choice == 2) {
        open_folder();
    }
    else if (user_choice == 3) {
        go_back_folder();
    }
    else if (user_choice == 4) {
        make_new_file();
    }
    else if (user_choice == 5) {
        remove_file();
    }
    else if (user_choice == 6) {
        copy_a_file();
    }
    else if (user_choice == 7) {
        search_file();
    }
    else if (user_choice == 8) {
        show_file_details();
    }
    else if (user_choice == 9) {
        cout << "\nThank you for using File Explorer!\n";
        cout << "Goodbye!\n\n";
        break;
    }
    else {
        cout << "ERROR: Invalid choice! Please enter 1-9.\n";
    }
}
```

```

    }

}

return 0;
}

```

Output:

Main Page

```

rakesh_raut@DESKTOP-LT108Q3:~$ cd file-explorer
rakesh_raut@DESKTOP-LT108Q3:~/file-explorer$ nano file_explorer.cpp
rakesh_raut@DESKTOP-LT108Q3:~/file-explorer$ g++ -o file_explorer file_explorer.cpp
rakesh_raut@DESKTOP-LT108Q3:~/file-explorer$ ./file_explorer

*** Welcome to File Explorer! ***
This program helps you manage files on your computer.

=====
FILE EXPLORER PROGRAM
=====
Current Location: .
=====
1. Show all files
2. Open a folder
3. Go back to previous folder
4. Create new file
5. Delete a file
6. Copy a file
7. Search for files
8. Show file details
9. Exit program
=====
Enter your choice: _

```

Test 1: Display Files (Option 1)

```

=====
Enter your choice: 1

--- Files in . ---
Name                Type      Size
-----
file_explorer       [FILE]    45072 bytes
file_explorer.cpp   [FILE]    8312 bytes
test_folder_new     [FOLDER]  4096 bytes
test_folder         [FILE]    21 bytes

=====
FILE EXPLORER PROGRAM
=====
Current Location: .

```

Test 2: Open a Folder (Option 2)

Test Case A: Valid Folder

```

=====
Enter your choice: 2
Enter folder name: test_folder_new
Moved to: ./test_folder_new

=====
FILE EXPLORER PROGRAM
=====
Current Location: ./test_folder_new
=====

```

Test Case B: Invalid Folder

```

=====
Enter your choice: 2
Enter folder name: nonexistent_folder
ERROR: Folder does not exist!
=====
FILE EXPLORER PROGRAM
=====
Current Location: ./test_folder_new
=====

```

Test 3: Go Back to Previous Folder

```

=====
Enter your choice: 3
Moved back to: .
=====
FILE EXPLORER PROGRAM
=====
Current Location: .
=====

```

Test 4: Create New File (Option 4)

```

=====
Enter your choice: 4
Enter new file name: test_report.txt
File created successfully!
Enter file content (type 'END' on a new line to finish):
This is line 1
This is line 2
This is line 3
END
File saved successfully!
=====
FILE EXPLORER PROGRAM
=====
Current Location: .
=====

```

Root directory after creating test_report.txt

```

Enter your choice: 1

--- Files in . ---
Name                Type      Size
-----
File_explorer       [FILE]    45072 bytes
File_explorer.cpp   [FILE]    8312 bytes
test_report.txt     [FILE]    45 bytes
test_folder_new     [FOLDER]  4096 bytes
test_folder         [FILE]    21 bytes

=====
FILE EXPLORER PROGRAM
=====
Current Location: .
=====

```

Test 5: Delete a File (Option 5)

Directory Structure before this operation

```

--- Files in . ---
Name                Type      Size
-----
File_explorer       [FILE]    45072 bytes
File_explorer.cpp   [FILE]    8312 bytes
test_report.txt     [FILE]    45 bytes
test_folder_new     [FOLDER]  4096 bytes
test_folder         [FILE]    21 bytes
delete_me.txt       [FILE]    4 bytes

=====
FILE EXPLORER PROGRAM
=====
Current Location: .
=====

```

Test Case A: Delete Existing File

```

=====
Enter your choice: 5
Enter file name to delete: delete_me.txt
File deleted successfully!
=====
FILE EXPLORER PROGRAM
=====
Current Location: .
=====

```

Directory after deletion:

```
=====
Enter your choice: 1
--- Files in . ---
Name                Type                Size
-----
file_explorer        [FILE]                45072 bytes
file_explorer.cpp    [FILE]                8312 bytes
test_report.txt      [FILE]                45 bytes
test_folder_new      [FOLDER]              4096 bytes
test_folder          [FILE]                21 bytes

=====
FILE EXPLORER PROGRAM
=====
Current Location: .
```

Test Case B: Delete Non-existent File

```
=====
Enter your choice: 5
Enter file name to delete: fake_file.txt
ERROR: Could not delete file!

=====
FILE EXPLORER PROGRAM
=====
Current Location: .
```

Test 6: Copy a File (Option 6)

Root Directory before this operation:

```
=====
Enter your choice: 1
--- Files in . ---
Name                Type                Size
-----
file_explorer        [FILE]                45072 bytes
file_explorer.cpp    [FILE]                8312 bytes
test_report.txt      [FILE]                45 bytes
test_folder_new      [FOLDER]              4096 bytes
test_folder          [FILE]                21 bytes
original.txt         [FILE]                29 bytes

=====
FILE EXPLORER PROGRAM
=====
Current Location: .
```

Content of original.txt

```
rakesh_raut@DESKTOP-LT10BQ3:~$ cd file-explorer
rakesh_raut@DESKTOP-LT10BQ3:~/file-explorer$ echo "Original content for testing" > original.txt
rakesh_raut@DESKTOP-LT10BQ3:~/file-explorer$ echo "Original content for testing" > original.txt
rakesh_raut@DESKTOP-LT10BQ3:~/file-explorer$ cat original.txt
Original content for testing
rakesh_raut@DESKTOP-LT10BQ3:~/file-explorer$ |
```

Test Case A: Successful Copy

```
=====
Enter your choice: 6
Enter source file name: original.txt
Enter destination file name: copy_of_original.txt
File copied successfully!

=====
FILE EXPLORER PROGRAM
=====
Current Location: .
```

Content of copy_of_original.txt after copy:

```
rakesh_raut@DESKTOP-LT10BQ3:~/file-explorer$ diff original.txt copy_of_original.txt
rakesh_raut@DESKTOP-LT10BQ3:~/file-explorer$ cat copy_of_original.txt
Original content for testing
rakesh_raut@DESKTOP-LT10BQ3:~/file-explorer$ |
```

Test Case B: Copy Non-existent File

```
=====
Enter your choice: 6
Enter source file name: doesnt_exist.txt
Enter destination file name: copy.txt
ERROR: Source file not found!

=====
FILE EXPLORER PROGRAM
=====
Current Location: .
```

Test 7: Search for Files (Option 7)

Directory before this operation

```
Enter your choice: 1

--- Files in . ---
Name                Type      Size
-----
file_explorer        [FILE]    45072 bytes
file_explorer.cpp     [FILE]    8312 bytes
notes.txt            [FILE]      0 bytes
reports              [FOLDER]  4096 bytes
test_report.txt       [FILE]     45 bytes
test_folder_new       [FOLDER]  4096 bytes
test_folder           [FOLDER]  4096 bytes
original.txt          [FILE]    29 bytes
copy_of_original.txt  [FILE]    29 bytes
report_2024.txt       [FILE]      0 bytes

=====
FILE EXPLORER PROGRAM
=====
```

Test Case A: Search with Matches

```
=====
Enter your choice: 7
Enter search term: report

Searching for files containing: report
-----
Found: reports [FOLDER]
Found: test_report.txt
Found: report_2024.txt

Total found: 3 file(s)

=====
FILE EXPLORER PROGRAM
=====
Current Location: .
=====
```

Test Case B: Search with No Matches

```
=====
Enter your choice: 7
Enter search term: xyz123notfound

Searching for files containing: xyz123notfound
-----
No files found!

=====
FILE EXPLORER PROGRAM
=====
Current Location: .
=====
```

Test Case C: Partial Match

```
Enter your choice: 7
Enter search term: test

Searching for files containing: test
-----
Found: test_report.txt
Found: test_folder_new [FOLDER]
Found: test_folder [FOLDER]

Total found: 3 file(s)

=====
FILE EXPLORER PROGRAM
=====
Current Location: .
```

Test 8: Show File Details (Option 8)

Test Case B: Existent File

```
=====
Enter your choice: 8
Enter file name: details_test.txt

--- File Information ---
File Name: details_test.txt
File Size: 30 bytes
Permissions: rw-
Last Modified: Sun Nov 9 09:42:55 2025

=====
FILE EXPLORER PROGRAM
=====
Current Location: .
=====
```

Test Case B: Non-existent File

```
=====
Enter your choice: 8
Enter file name: notfound.txt
ERROR: File not found!

=====
FILE EXPLORER PROGRAM
=====
Current Location: .
=====
```

Test 9: Exit Program (Option 9)

```
=====
Enter your choice: 9
Thank you for using File Explorer!
Goodbye!
nakesh_raut@DESKTOP-LT10RQ3:~/file-explorer$
```

. CONCLUSION

This File Explorer Application successfully demonstrates comprehensive file management capabilities through a console-based interface. The project achieves all its primary objectives:

- ✓ **System Integration**
Successfully interfaces with Linux OS using POSIX APIs and system calls for file operations.
- ✓ **Core Functionality**
Implements all essential file operations: create, read, update, delete, copy, and search with robust error handling.
- ✓ **User Experience**
Provides an intuitive menu-driven interface with clear feedback and error messages.
- ✓ **Code Quality**
Follows modular design principles with well-organized functions, proper error handling, and input validation.
- ✓ **Performance**
Efficiently handles file operations with minimal resource usage and fast response times.

Github Repo link: <https://github.com/Rex1671/CapStoneProject>