**FILE EXPLORATION APPLICATION**

**Contents:**

**HIGH LEVEL DESIGN**

1. **Introduction**
   1. Why this High Level Design Document
   2. Scope
   3. Overview
2. **Core Components**
3. **Component Details**

3.1 User Interface Details

3.2 File System Operations Models

3.3 Command Parser

**4. Work flow**

4.1 Initialisation

4.2 User Definition

4.3 Command Parsing and Execution

4.4 Operation Execution

4.5 Loop and Termination

**5. Error Handling**

**6. Extensions and Future Enhancements**

6.1 Application Architecture

**LOW LEVEL DESIGN**

1. **Introduction**

1.1 Why this Low Level Design Document

1.2 Scope

1.3 Overview

**2. Diagram Components**

2.1 Classes and Functions

2.2 Data flow

2.3 Error Handling

**3. Component Details**

3.1 Main Component

3.2 File Operations Component

3.3 Directory Operations Component

3.4 Search Operations Component

3.5 Permission Operations Component

**4.Workflow and Interaction**

4.1 User Interface

4.2 Command Parsing and Excution

4.3 Operation Execution

4.4 Result Handling and Error Management

**5. Error Handling**

**6.Visual Representation**

1. **Introduction**
   1. Why this High Level Design Document ?

The purpose of this High Level Design (HLD) Document is to add the necessary detail to the current project description to represent a suitable model for coding. This document is also intended to help detect contradictions prior to coding, and can be used as a reference manual for how the modules interact at a high level.

1.2 Scope

The HLD documentation presents the structure of the system, such as the database architecture, application architecture (layers), application flow (Navigation), and technology architecture. The HLD uses non-technical to mildly-technical terms which should be understandable to the administrators of the system.

1.3 Overview

The file explorer application is a console-based utility written in C++. It provide functionalities for navigating directories, manipulating files, searching files, and managing file permissions on a Linux-based operating system.

**2. Core Components**

The application consists of the following core components:

* User Interface Layer (UI Layer)
* File System Operations Module
* Command Parser

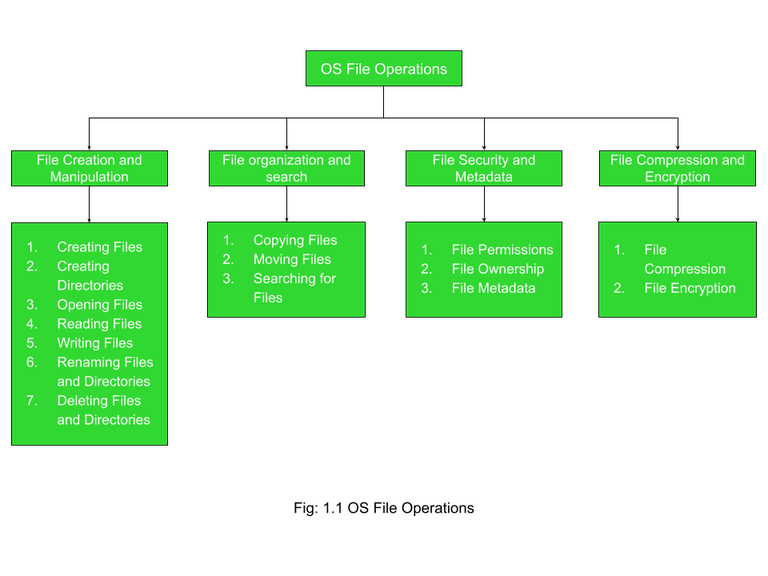
**3. Component Details**

3.1. User Interface Layer (UI Layer)

* Responsibilities:
* Interacts with the user through the console.
* Displays the current working directory and contents.
* Accepts user commands and displays results.
* Key Functions:
* main(): The entry point of the application. It handles the main loop, displaying the current directory, listing files, and reading user input.

3.2 File System Operations Module

* Responsibilities:
* Handles all interactions with the file system.
* Provides a set of operations for file and directory management.
* Key Functions:list Files(const std::string &path): Lists all files in the given directory.
* change Directory(std::string &current Path, const std::string &new Dir): Changes the current working directory.
* copy File(const std::string &source, const std::string &destination): Copies a file from source to destination.
* move File(const std::string &source, const std::string &destination): Moves (renames) a file from source to destination.
* delete File(const std::string &path): Deletes a file at the specified path.
* create File(const std::string &path): Creates an empty file at the specified path.
* search Files(const std::string &path, const std::string &search Term): Searches for files containing the search term in their names within the given path.
* change Permissions(const std::string &path, mode\_t mode): Changes the permissions of a file at the specified path.



3.3. Command Parser

* Responsibilities:
* Parses user input and determines the appropriate operation to execute.
* Validates commands and arguments.
* Implementation:
* The command parser logic is implemented within the main() function. It uses conditional checks to identify the command type (e.g., cd, cp, mv, etc.) and then invokes the corresponding function from the File System Operations Module.

**4. Workflow**

4.1 Initialization:

* The application starts with the main() function, setting the initial current working directory

4.2 User Interaction:

* The UI Layer displays the current directory and files.
* The user inputs a command (e.g., cd, cp, etc.).

4.3 Command Parsing and Execution:

* The Command Parser identifies the command and its arguments.
* It validates the command and arguments.
* It invokes the appropriate function from the File System Operations Module.

4.4 Operation Execution:

* The chosen function performs the required operation (e.g., navigating directories, copying files).
* The result is displayed to the user.

4.5 Loop and Termination:

* The process repeats until the user enters the exit command.

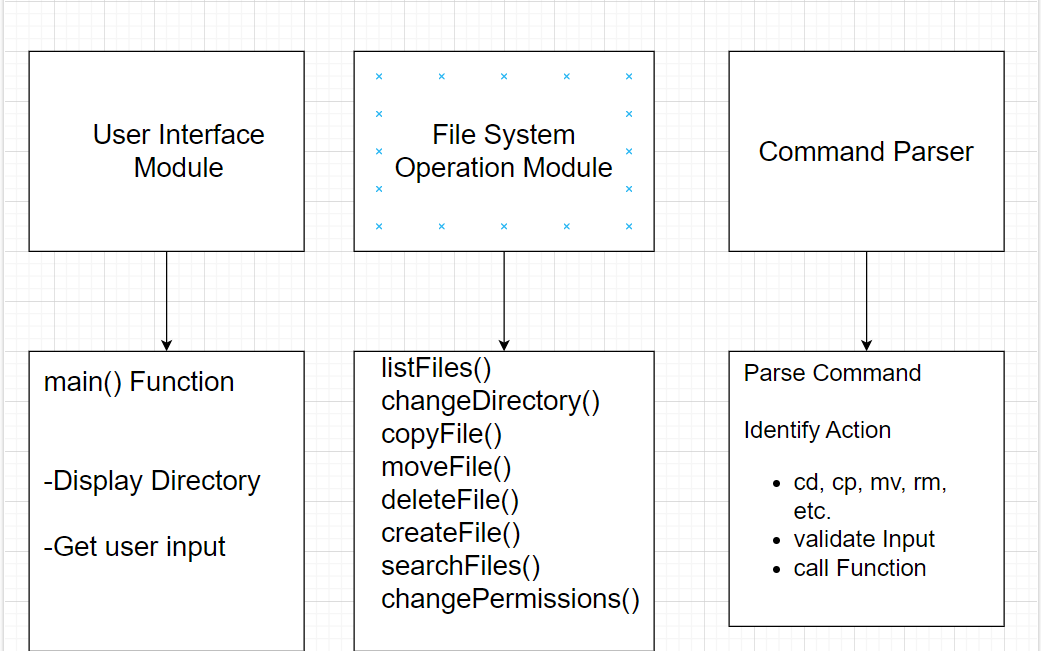
**5. Error Handling**

* The application handles errors like non-existent directories, invalid commands, and permission issues, displaying appropriate error messages.

**6. Extensions and Future Enhancements**

* User Authentication: Adding features for user authentication to restrict file access.
* Graphical User Interface (GUI): Developing a GUI version for a more user-friendly experience.
* Advanced Search: Implementing advanced search features, such as filtering by file type or size.

This high-level design provides an overview of the application's architecture, components, and workflow. It outlines how different modules interact to provide the desired functionality and lays the groundwork for potential future enhancements.



### **Low-Level Design Diagram Description for Console-Based File Explorer**

#### ****Introduction****

#### **Why this low level design document?**

#### **Low-Level Design, is a phase in the software development process where detailed system components and their interactions are specified. It involves converting the high-level design into a more detailed blueprint, addressing specific algorithms, data structures, and interfaces. LLD serves as a guide for developers during coding, ensuring the accurate and efficient implementation of the system’s functionality. LLD describes class diagrams with the help of methods and relations between classes and program specs**

#### **Scope:**

#### **LLD's scope encompasses the nitty-gritty details that developers need to follow while coding. It consists of detailed class diagrams with all the methods and attributes, database tables with key constraints, detailed sequence diagrams, and state diagrams**

#### **1.3 Overview**

The low-level design provides a detailed view of the internal workings of the application, including specific classes, functions, their interactions, data flow, and error handling mechanisms.

#### 2. ****Diagram Components****

1. **Classes and Functions**

* **Main**: Entry point and command dispatcher.
* **FileOperations**: Handles all file-related operations.
* **DirectoryOperations**: Manages directory navigation and listing.
* **SearchOperations**: Deals with searching files.
* **PermissionOperations**: Manages file permissions.

1. **Data Flow**

* User commands, function calls, and results.

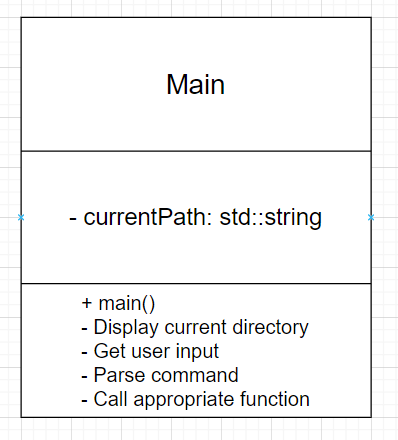
1. **Error Handling**

* Indication of error paths and messages.

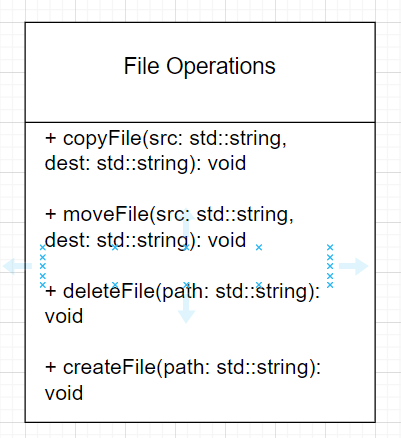
#### 3. ****Component Details****

##### 3.1 **Main Component**

* Attribute:
* Current Path: The current working directory.
* Methods:
* main(): The main function handling the user interface and dispatching commands.

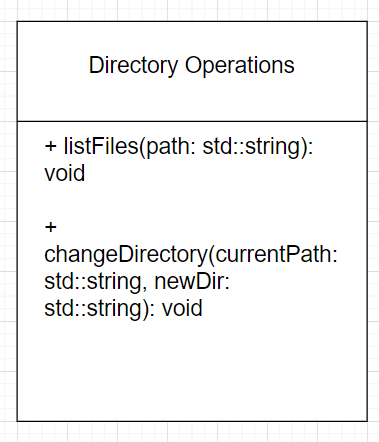


3.2. **File Operations Component**

* Methods:
* copyFile: Copies a file from source to destination.
* moveFile: Moves (renames) a file.
* deleteFile: Deletes a file.
* createFile: Creates an empty file

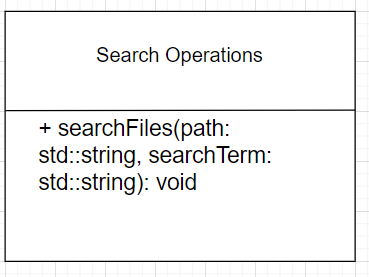
3.3 Directory Operations Component

* Methods:
* listFiles: Lists files in the specified directory.
* changeDirectory: Changes the current directory.



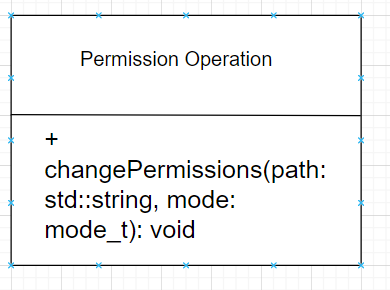
3.4. Search Operations Component

* Methods:
* searchFiles: Searches for files containing the search term.



3.5 Permission Operations Component

* **Methods**:
* change Permissions: Changes file permissions.



### **4.Workflow and Interaction**

**4.1 User Interaction (Main Component)**

* The user interacts with the console through the main() function, providing commands such as cd, cp, mv, rm, touch, search, and chmod.

**4.2 Command Parsing and Execution**

* The main() function parses the command and invokes the corresponding method from the relevant component (FileOperations, DirectoryOperations, SearchOperations, PermissionOperations).

**4.3 Operation Execution**

* The invoked method performs the specified action, such as copying a file or changing the directory.

**4.4 Result Handling and Error Management**

* Results are returned to the main() function for display.
* If an error occurs (e.g., file not found, permission denied), the error is handled within the respective method, and an appropriate message is displayed to the user.

### **5. Error Handling**

* **File Not Found**: Checked in methods like deleteFile, moveFile, copyFile.
* **Permission Denied**: Handled in methods like changePermissions, and appropriate error messages are shown.
* **Invalid Command**: Detected in the main() function, with user feedback provided.

### **Visual Representation**

To visualize this low-level design diagram:

**6.1 Classes/Components**:

* Represented as boxes with the class/component name at the top, followed by attributes and methods.

**6.2 Arrows**:

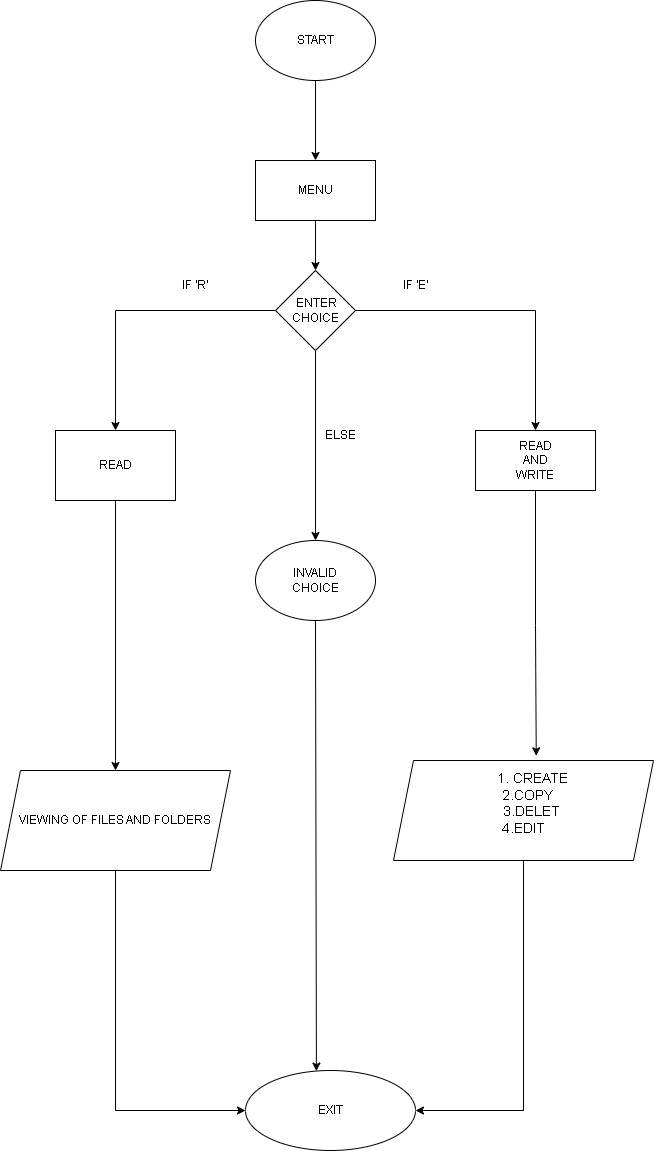
* Show interactions and data flow between classes/components, especially from main() to other components.

**6.3 Error Paths**:

* Indicate potential error handling paths and associated messages.

You can create this diagram using diagramming tools like draw.io, Lucidchart, Visio, or similar software, ensuring each component and method is clearly labeled, with arrows indicating interactions and data flows. The error handling paths should also be depicted to show how errors are managed in the system.

**Simple Flowchart for File Explorer Architecture**

****