**File Explorer Application Requirements Document**

**Project Title:** File Explorer Application

**Objective:**  
Develop a console-based file explorer application in C++ that interfaces with the Linux operating system to manage files and directories. The application will support listing files, navigating directories, manipulating files (create, copy, move, delete), searching for files, and managing file permissions. Additionally, the application will implement logging to capture all actions performed by the user, with separate log files for both server and client components.

**1. Functional Requirements**

**1.1. File and Directory Operations**

1. **List Directory Contents**
   * **Description:** The application must display the contents of the current directory, including files and subdirectories.
   * **Input:** Directory path (optional; if not provided, the current directory will be used).
   * **Output:** A list of files and directories within the specified directory.
2. **Change Directory**
   * **Description:** The user must be able to navigate between directories.
   * **Input:** The path to the target directory.
   * **Output:** The current working directory is updated to the specified directory.
3. **Create File**
   * **Description:** The application should allow users to create a new, empty file in a specified directory.
   * **Input:** Full file path including the file name.
   * **Output:** A new file is created at the specified location.
4. **Create Directory**
   * **Description:** Users should be able to create a new directory.
   * **Input:** Full path to the new directory.
   * **Output:** A new directory is created at the specified location.
5. **Delete File/Directory**
   * **Description:** The application must support deleting files and directories.
   * **Input:** Full path to the file or directory.
   * **Output:** The specified file or directory is deleted.
6. **Copy File**
   * **Description:** Users should be able to copy a file from one location to another.
   * **Input:** Source file path and destination path.
   * **Output:** The file is duplicated at the destination path.
7. **Move File**
   * **Description:** The application should allow users to move a file from one location to another.
   * **Input:** Source file path and destination path.
   * **Output:** The file is moved to the destination path.
8. **Search File**
   * **Description:** Users should be able to search for a file by name within the current directory.
   * **Input:** File name or partial file name.
   * **Output:** List of matching files in the current directory.
9. **Change File Permissions**
   * **Description:** Users should be able to modify the permissions (read, write, execute) of a file or directory.
   * **Input:** File/directory path and new permissions.
   * **Output:** The permissions of the specified file or directory are updated.

**1.2. Logging**

1. **Log File Creation**
   * **Description:** The application should create and maintain separate log files for both server and client components.
   * **Input:** Log entries (actions performed).
   * **Output:** Logs are written to the console and respective log files.
2. **Log Entries**
   * **Description:** Every action performed by the user should be logged with a timestamp.
   * **Input:** Action details (e.g., file creation, deletion).
   * **Output:** Entry in the log file with details about the action.

**1.3. User Interface**

1. **Command-Line Interface (CLI)**
   * **Description:** The application must provide a text-based interface where users can input commands to perform operations.
   * **Input:** User commands (e.g., ls, cd, createfile).
   * **Output:** Feedback in the form of command execution results.
2. **Menu Display**
   * **Description:** The application should display a menu with available commands to guide users.
   * **Input:** User interaction to choose a command.
   * **Output:** Execution of the selected command.

**1.4. Error Handling**

1. **Invalid Commands**
   * **Description:** The application should handle invalid or unrecognized commands gracefully by displaying an appropriate error message.
   * **Input:** Invalid command.
   * **Output:** Error message.
2. **File/Directory Not Found**
   * **Description:** The application must check for the existence of files and directories before performing operations and handle cases where they do not exist.
   * **Input:** Non-existent file or directory path.
   * **Output:** Error message indicating that the file or directory does not exist.
3. **Permission Denied**
   * **Description:** If the user does not have the required permissions to perform an action, the application should notify the user.
   * **Input:** Attempt to access a file/directory without appropriate permissions.
   * **Output:** Error message indicating permission denial.

**2. Non-Functional Requirements**

1. **Performance**
   * The application should perform file operations (listing, creation, deletion, etc.) within a reasonable time, even for directories containing a large number of files.
2. **Scalability**
   * The application should be able to handle directories with a large number of files and deep directory hierarchies.
3. **Security**
   * The application should ensure that only users with appropriate permissions can modify files or directories.
4. **Usability**
   * The command-line interface should be intuitive and easy to use, with clear error messages and feedback.
5. **Portability**
   * The application should be able to run on any Linux distribution with the necessary C++ development environment installed.

**3. System Requirements**

**3.1. Software Requirements**

* **Operating System:** Linux (Ubuntu 20.04 or later recommended)
* **Development Environment:** GCC/G++ compiler, Make utility
* **Libraries:** Standard C++ libraries (iostream, fstream, sys/stat.h, dirent.h, etc.)

**3.2. Hardware Requirements**

* **Processor:** Any modern CPU supporting Linux
* **RAM:** Minimum 512 MB (1 GB recommended)
* **Disk Space:** 100 MB for application and logs

**4. Implementation Details**

**4.1. Development Environment Setup**

* **Install GCC/G++ Compiler:** sudo apt-get install build-essential
* **Install Make:** sudo apt-get install make

**4.2. Directory Structure**

* **/src:** Source files (.cpp)
* **/include:** Header files (.h)
* **/bin:** Compiled binaries
* **/logs:** Log files

**4.3. Makefile**

* **Targets:**
  + all: Compile the application.
  + clean: Remove compiled binaries and logs.
  + run: Execute the application.

**5. Testing Requirements**

* **Unit Tests:** Test individual functions (e.g., listDirectory, createFile).
* **Integration Tests:** Test the application’s ability to perform sequences of operations (e.g., navigate to a directory, create a file, and then delete it).
* **System Tests:** Test the application in a real-world environment with large directories and multiple users.
* **Logging Tests:** Ensure that logs are correctly created and updated for both server and client components.

**6. Deployment and Maintenance**

* **Deployment:**
  + The application should be deployed on a Linux system using the compiled binaries.
  + Logs should be monitored regularly to ensure the application is functioning correctly.
* **Maintenance:**
  + Regular updates may be needed to improve performance or add new features.
  + Bug fixes should be applied promptly as they are identified.

**7. Appendix**

* **References:**
  + Linux File System Hierarchy: https://linux.die.net/man/7/hierarchy
  + C++ Standard Library Documentation: https://en.cppreference.com/w/

This requirements document provides a comprehensive overview of the File Explorer Application, detailing its functional and non-functional requirements, system needs, and implementation strategies.