***Assignment 4***

***Subject:***

*Operating System*

***Topic:***

*Project Document*

***Submitted By:***

*Aisha(G1F22UBSCS254)*

*Eman(G1F22UBSCS255)*

***Submitted To:***

*Prof.Hassan Rauf*

***Submission Date:***

*27-02-2025*

***Introduction:***

A **File Management System (FMS)** is a software application that enables users to create, organize, store, retrieve, and manage files on a computer or a server. It serves as an essential component of computer operating systems, providing an interface for users to interact with their data efficiently.

## ***Purpose***

The File Management System (FMS) in the code is designed to manage files through a command-line interface, enabling users to:

* Create, read, write, delete, rename, move, and copy files.
* Retrieve file information, such as size and last modified date.
* Ensure secure access through user authentication

## Objectives

* **User-Friendly**: Simplifies file operations for users.
* **Security**: Protects data with authentication.
* **Comprehensive Management**: Covers all essential file operations.
* **Efficiency**: Enhances productivity in file handling.
* **Quick Information Access**: Provides easy retrieval of file details.

Importance:

**Data Protection**: Safeguards sensitive information from unauthorized access.

**Scalability**: Adapts to growing file storage needs efficiently.

## Project Scope

The File Management System (FMS) is designed to provide a comprehensive solution for managing files on a computer through a command-line interface. The system will cater to essential file operations, ensuring ease of use, security, and efficient organization of files. The project encompasses the creation, modification, retrieval, and deletion of files, alongside user authentication to safeguard access.

## Features and Functionalities

**1-User Authentication**:

* 1. Secure login system to ensure only authorized users can access file management functions.

**2-File Operations:**

* 1. **Create File**: Ability to create new files.
  2. **Read File**: Retrieve and display content of existing files.
  3. **Write to File**: Append content to existing files.
  4. **Delete File**: Remove files from the system.
  5. **Rename File**: Change the name of existing files.
  6. **Move File**: Relocate files to a new directory.
  7. **Copy File**: Duplicate files to a specified location.

**3-File Information Retrieval**:

* 1. Display details such as file size and last modified date.

**4-File Listing**:

* 1. List all files in the current directory for easy navigation.

**5-File Search**:

* 1. Search for files based on a specified pattern.

## Target Users

* **Individual Users**: People looking for a simple tool to manage personal files, such as documents, photos, and media.
* **Business Professionals**: Employees and managers who need to organize and access business documents efficiently.
* **Teams and Collaborators**: Groups that require shared access to files, facilitating teamwork and project management.
* **System Administrators**: IT professionals who oversee file management and security within organizations.

# Technology Stack for the File Management System

## Programming Languages Used

* **C++**: The primary programming language utilized for developing the File Management System. C++ is chosen for its performance, object-oriented features, and standard library support for file handling and system operations.

## Development Environment:

* **Dev-C++**: The chosen integrated development environment (IDE) for this project. Dev-C++ provides a user-friendly interface and supports various compilers, making it ideal for C++ development.

## Libraries and Frameworks

* **Standard Library**:
* **<iostream>**: Used for input and output operations.
* **<fstream>**: Provides functionalities for file input and output.
* **<filesystem>**: Introduced in C++17, this library is used for file system operations such as file manipulation and directory traversal.
* **<string>**: Used for handling string data.
* **<unordered\_map>**: Used for implementing the user authentication system, allowing efficient storage and retrieval of username-password pairs.
* **<chrono>**: Used to handle time-related operations, such as retrieving the last modified time of files.

Concepts Include:

## 1. File System Management

The File Management System primarily interacts with the file system of the operating system. Concepts utilized include:

* **File Creation and Deletion**: The system allows users to create and delete files, directly interfacing with the underlying file system.
* **File Manipulation**: Functions such as renaming, moving, and copying files leverage file system operations to manage file organization and structure.
* **File Information Retrieval**: The system retrieves metadata about files, such as size and last modified date, using file system calls.

## 2. Device Management (I/O Handling)

* **Input/Output Operations**: The system uses file streams (<fstream>) for reading from and writing to files, which involves I/O management by the operating system. This includes buffering, caching, and ensuring efficient data transfer between the application and storage devices.

## 3. Security and Access Control

* **User Authentication**: The system implements a basic authentication mechanism using a username-password pair stored in an unordered\_map. This ensures that only authorized users can access file management functionalities, thereby enforcing security measures.

## 4. Memory Management

While the code does not directly implement advanced memory management concepts like paging or segmentation, it relies on the operating system's memory management to handle:

* **Dynamic Memory Allocation**: The program allocates memory for strings and file streams as needed, managed by the operating system.

## 5. Process Management

The code does not explicitly deal with process management concepts such as scheduling or threading. However, the following can be inferred:

* **Single-threaded Execution**: The system operates in a single-threaded manner, executing commands sequentially. If extended, multithreading could be incorporated to handle multiple file operations concurrently.

## Core Algorithms and Logic

* **User Authentication**: Uses an unordered map to store username-password pairs and checks credentials for access.
* **File Operations**: Supports creating, reading, writing, deleting, renaming, and moving files through dedicated functions.
* **File Information Retrieval**: Retrieves file metadata such as size and last modified date using standard library functions.

## Module Interactions

* **User Authentication Module**: Handles login/logout processes and verifies credentials with the user interface.
* **File Operations Module**: Manages all file manipulations, checking user authentication before allowing actions.
* **User Interface Module**: Collects user input and displays output, directing commands to the appropriate modules.

Conclusion:

## Summary of Project Outcomes

* **Functional File Management**: Successfully implemented a system that allows users to perform essential file operations, including creating, reading, writing, and deleting files.
* **User Authentication**: Developed a secure method for user authentication, ensuring that only authorized users can access file management functionalities.
* **Modular Design**: Achieved a well-structured codebase with clear separation of concerns, facilitating easier maintenance and scalability.

## Key Takeaways and Learnings

* **Importance of Security**: Understanding the significance of user authentication in protecting sensitive file operations.
* **Modular Programming**: Gained insights into the benefits of modular design in improving code readability, maintainability, and collaboration.
* **Error Handling**: Learned to implement basic error handling to enhance user experience and prevent program crashes.
* **C++ Proficiency**: Improved skills in C++ programming, especially in file handling and standard library usage.
* **Development Environment Familiarity**: Gained experience with Dev-C++ as an IDE, enhancing efficiency in coding and debugging.