A SYNOPSIS ON

FILE MANAGEMENT SYSTEM

Submitted in partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

In

Computer Science & Engineering

Submitted by:

Rahul Singh Dhami	**2261456**
Chandu Singh	**2261151**
Pawan Singh Mehra	**2261414**
Varun Mehta	**2261593**

Under the Guidance of

Mr Ansh Dhingra

Lecturer Of Department of CSE

Project Team ID: 59

Department of Computer Science & Engineering

Graphic Era Hill University, Bhimtal, Uttarakhand

March-2025

Graphic Era Hill University
BHIMTAL CAMPUS

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the Synopsis entitled **"File Management System"** in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science & Engineering of the Graphic Era Hill University, Bhimtal campus and shall be carried out by the undersigned under the supervision of **Mr Ansh Dhingra , Lecturer Of Dep. of CSE**, Department of Computer Science & Engineering, Graphic Era Hill University, Bhimtal.

Rahul Singh Dhami	2261456	signature
Chandu Singh	2261151	signature
Pawan Singh Mehra	2261414	signature
Varun Mehta	2261593	signature

The above mentioned students shall be working under the supervision of the undersigned on the **"File Management System"**

Signature	Signature
Mr Ansh Dhingra	**Head of the Department**

<u>Internal Evaluation (By DPRC Committee)</u>

Status of the Synopsis: Accepted / Rejected

Any Comments:

Name of the Committee Members: **Signature with Date**

1.

2.

Table of Contents

Chapter 1

Introduction and Problem Statement

1. Introduction

In today's digitally driven world, effective file management is not merely a convenience—it is a cornerstone for productivity and organization. Managing and accessing files efficiently supports smooth daily operations, whether for personal data or large-scale business operations. As digital data continues to grow exponentially, robust file management systems are becoming increasingly indispensable. They empower users to effortlessly create, modify, and archive various file types while ensuring their data remains secure and easy to retrieve, which is a critical capability in any modern computing environment.

Our project envisions a sophisticated File Management System web application that pushes beyond the boundaries of traditional solutions. It is designed to provide an all-encompassing platform where users can create, delete, modify, and upload files with complete ease. Furthermore, the application is tailored to deliver enhanced user experiences through integrated multimedia and document viewing functionalities. With features such as a built-in image viewer, music and video players, and multiple document viewers (supporting formats like PDF, Word, Excel, and more), the system is set to redefine the way users interact with their digital content.

From a technical standpoint, the development of this project demonstrates a fusion of web technologies and core operating system concepts. On the client side, the interface is constructed using HTML, CSS, and JavaScript with Bootstrap, ensuring a responsive and visually appealing design. The server side leverages the power of Node.js with Express.js for efficient request handling, while MongoDB acts as our choice for a scalable and flexible database solution. In addition, by integrating Google Cloud services for data storage, our application aligns itself with proven cloud architectures that provide reliability and security comparable to platforms like Google Drive.

Another distinguishing aspect of our project is its commitment to incorporating critical operating system principles. By implementing file permission controls, multithreading techniques for enhanced performance, and inter-process communication (IPC) protocols, our system mirrors functionalities typically reserved for lower-level operating systems. This not only enriches the application's robustness but also presents a valuable learning opportunity in blending high-level web development with system-level programming concepts.

In essence, this File Management System is more than a mere digital filing cabinet—it is a dynamic, multi-functional platform designed to meet the multifaceted needs of modern users. It combines the practical utility of a file management tool with the versatility of multimedia processing and the rigor of operating system principles, all underpinned by modern web development technologies. This project embodies a forward-thinking approach, addressing current challenges while laying a foundation for scalable, secure, and user-friendly file management in the digital era.

Expanding on these themes, the following sections of this synopsis will delve into our project's objectives, detailed system architecture, and the innovative methods we have adopted to integrate traditional operating system features with modern web technologies.

0. Problem Statement

Key Challenges in Development:

1. Security and File Permissions: Designing a system that allows file creation, deletion, and modification means you must establish robust security measures. One major problem is ensuring that users only have access to files they're permitted to view or edit. Implementing file permission systems—similar to those found in traditional operating systems—forces you to design a granular access control mechanism. This involves not only verifying user identities but also managing different permission levels to prevent unauthorized data access or unintended data breaches.

2. Concurrency and Multi-threading: With features like simultaneous file uploads, real-time modifications, and multi-user access, handling multiple operations concurrently is a significant challenge. Multi-threading and proper synchronization mechanisms become essential to avoid data corruption, deadlocks, or race conditions. Consequently, the system must be designed to handle parallel processing efficiently with a consistent performance, ensuring that the system remains responsive even under heavy loads.

3. Data Consistency and Integrity: Managing files across various operations (creation, deletion, uploads, etc.) in a distributed environment (such as using Google Cloud storage) requires careful planning around data consistency. This introduces challenges related to data redundancy, backup, and recovery especially when considering options like version control to track changes in files, ensuring that user data is never lost even if errors occur during transmission or storage.

4. User Interface and Experience: Balancing a robust back-end with a user-friendly front-end is vital. While technical features are critical, the interface must be intuitive. Users expect quick, reliable access to their files and previews, whether they're playing a video, viewing a PDF, or editing a document. Handling various file types smoothly and providing integrated viewers for images, videos, music, and documents means that each viewer must be optimized for its corresponding file format while still being accessible on a responsive website.

5. Performance and Scalability: As the application scales with more users and larger file sizes, ensuring quick response times and efficient data processing becomes a significant challenge. This requires that the system is rigorously tested for scalability. Integrating cloud services such as Google Cloud not only provides scalability but also brings further challenges like managing network latency and ensuring the reliability of data transfers.

5. **Inter-process Communication (IPC):** Incorporating IPC within the system architecture is another challenge. Effective inter-process communication is crucial to ensure that separate system components (like the file system module, multimedia viewers, and user authentication module) can communicate seamlessly without causing performance bottlenecks or data inconsistencies.

How the Project Solves Everyday Problems

Streamlined File Management: By integrating core file operations—such as creation, deletion, modification, and upload—in one application, the project offers a centralized solution for file organization. Users can filter, search, and manage their files swiftly, reducing the time spent on locating documents and organizing data. This is particularly valuable for individuals juggling multiple file formats and large amounts of data.

Enhanced Security and Data Privacy: With robust file permission systems and secure cloud storage, every user's data is safeguarded against unauthorized access. This approach instills confidence and peace of mind, ensuring personal and business files are protected—an everyday concern for all users in a digital age.

Multimedia and Document Accessibility: The built-in image, music, video, and document viewers eliminate the need to switch between different applications. This not only saves time but also enhances productivity by allowing users to preview and work on their files directly within the system. The integrated approach replicates the convenience of platforms like Google Drive while adding additional functionalities.

Performance Under Pressure: By incorporating multi-threading and IPC, the system is built to manage simultaneous requests and operations without lag. This means even during periods of heavy usage, users can expect a responsive system that prioritizes efficiency and smooth data handling.

Scalability and Reliability: Leveraging cloud-based storage ensures that the system can grow alongside its user base. The scalable nature of cloud services means that as more users join and more data is processed, the system can adapt without significant downtime or performance issues—making it a future-proof solution that evolves with user needs.

Chapter 2

Background/ Literature Survey

In the present times, research work is being carried out in the intersection of file management systems, operating system concepts, and web-based applications. As digital data repositories grow exponentially, robust strategies for managing, organizing, and retrieving files have become indispensable. Early studies on file systems focused primarily on hierarchical organization and basic file operations. Over time, research evolved to address issues such as data consistency, security, and concurrency control which are essential in modern multi-user and cloud-based environments.

Recent literature has explored the integration of traditional file system methods with contemporary web technologies. Numerous studies have presented techniques for improving access speeds and handling simultaneous user operations through multi-threading and inter-process communication (IPC). For example, research into distributed file systems, such as the Google File System and Hadoop Distributed File System, demonstrates how leveraging multiple nodes and concurrent processes can lead to scalable and resilient architectures. These advances have directly influenced modern web applications that aim to offer functionalities similar to large cloud drives, ensuring efficient file storage and retrieval even under heavy use.

Another significant area of investigation deals with multimedia integration within file management systems. With users increasingly relying on digital media, research has extended into the design of integrated viewers for images, audio, video, and various document types. Studies in UI/UX design emphasize the necessity for responsive, seamless interfaces that allow users to interact with heterogeneous data formats without compromising on performance or ease of use. The introduction of dedicated document viewers for PDFs, Word files, and spreadsheets underlines the trend towards unified platforms that handle both conventional file management and multimedia content viewing.

Security is yet another critical facet addressed by contemporary research. The design of file management systems now increasingly incorporates advanced security protocols, including multi-level file permissions and encryption standards. Such measures are vital for preventing unauthorized access and ensuring data integrity, especially in applications that deal with sensitive personal or organizational information. Academic papers in this field advocate for granular access control mechanisms and robust backup strategies, which have been adapted and optimized for web-based systems that rely on cloud storage solutions.

Furthermore, literature on the fusion of operating system (OS) concepts with web development underscores the benefits of embedding system-level functionalities directly into online applications. By leveraging OS principles—such as process synchronization, multi-threading, and IPC—researchers have demonstrated that web applications can achieve higher reliability, fault tolerance, and efficiency. These insights are particularly relevant in building file management systems that not only offer basic file handling operations but also mimic the controlled environment of operating systems, providing users with a secure and high-performance platform.

In the realm of securing digital data, William Stallings (2013) in Cryptography and Network Security: Principles and Practice (6th Edition) offers an exhaustive treatment of encryption methods, authentication schemes, and access control mechanisms critical to protecting file systems. His work details the implementation of cryptographic protocols and network security practices that safeguard data integrity and privacy—a fundamental aspect for any modern file management system, particularly those deployed in the cloud

As files increasingly include multimedia content, specialized file system paradigms have emerged to handle the real-time requirements of audio and video streaming. Academic studies—such as those encapsulated in the lecture notes on multimedia file system paradigms (Chen & Zhu, n.d.) available from university course resources—have explored push-based data delivery models that ensure uninterrupted media streaming. These paradigms address challenges such as precise timing of data delivery and resource reservation, thereby enhancing user experience when accessing multimedia files.

Large-Scale Data Processing Dean and Ghemawat (2004) in MapReduce: Simplified Data Processing on Large Clusters presented a transformative programming model for processing enormous volumes of data through parallelization. Their MapReduce paradigm abstracts the complexity of data partitioning, fault tolerance, and load balancing—insights that are directly applicable to file management systems tasked with handling concurrent operations across distributed nodes. This work has influenced the way modern systems process and manage massive datasets efficiently.

Distributed File Systems Ghemawat, Gobioff, and Leung (2003) introduced the Google File System—a pioneering work that tackled the challenges of building a scalable and fault-tolerant distributed file system. In The Google File System, the authors explained innovative strategies like data replication, chunk-based storage, and streamlined recovery mechanisms designed to operate efficiently on clusters of commodity hardware. Their contributions have laid the groundwork for many modern cloud-based file management solutions that require robust scalability and high availability.

Chapter 3

Objectives

1. **Design a Robust, User-Friendly File Management Platform** Develop an intuitive and efficient web application that enables users to execute core file operations such as creation, deletion, modification, and uploading with ease. By leveraging standard front-end technologies like HTML, CSS, JavaScript, and Bootstrap, the system will offer a clean, responsive interface. This objective is focused on ensuring that even non-technical users can navigate and manage their files seamlessly, thereby reducing the learning curve and enhancing productivity.

2. **Integrate Comprehensive Multimedia and Document Viewing Capabilities** Incorporate integrated viewers for a variety of file formats, including images, audio, video, as well as documents like PDF, Word, and Excel files. The goal is to ensure that users can preview and interact with their files without the need for multiple external applications. This functionality will enrich the user experience and promote a unified approach to managing multimedia content and documents, saving time and reducing system clutter.

3. **Implement Advanced Operating System Concepts within a Web Environment** Utilize core operating system principles—such as finely grained file permission controls, multi-threading for optimal task handling, and inter-process communication (IPC) for seamless coordination among system components. Integrating these OS concepts will not only enhance the security and stability of the application but will also ensure efficient performance, particularly when handling simultaneous operations or high user traffic in a distributed environment.

4. **Leverage Cloud-Based Storage for Scalability and Enhanced Data Security** Integrate Google Cloud services to manage the storage and backup of user data, similar to established platforms like Google Drive. This objective emphasizes the deployment of robust cloud storage solutions that guarantee data integrity, facilitate quick recovery, and provide scalability as user demands grow. By adopting cloud technology, the project aims to offer a highly reliable system that meets both current and future storage requirements with state-of-the-art security measures.

Together, these objectives create a comprehensive roadmap for the proposed file management system. The focus on usability, multimedia integration, advanced OS concepts, and cloud storage ensures that the project addresses the pressing needs of modern users—delivering a secure, scalable, and highly efficient solution for everyday file management challenges.

Chapter 4

Hardware and Software Requirements

4.1 Hardware Requirements

Sl. No	Name of the Hardware	Specification
1	Processor	Amd or Intel core i3/i5 (or higher)
2	RAM	Minimum 4Gb Or Higher
3	Storage	Not Ncessary Required
4	Network	High Speed Internet Connection

4.2 Software Requirements

Sl. No	Name of the Software	Specification
1	Opreating System	Windows 11
2	Code Editor	Visual Studio Code
3	Browser	Microsoft Edge / Chrome

Chapter 5

Possible Approach/ Algorithms

1. Overview and System Architecture

The proposed file management system is designed as an end-to-end web application adhering to a modular, service-oriented architecture. The project leverages the Model-View-Controller (MVC) pattern to separate concerns:

Model: Manages file data, metadata, and user permission structures using a NoSQL database (MongoDB) that efficiently supports document-oriented storage and flexible data schemas.

View: Utilizes HTML, CSS, JavaScript, and Bootstrap to create a responsive, user-friendly interface that is adaptable across various devices.

Controller: Handles asynchronous operations, file transactions, and data synchronization between the client and server using Node.js with Express.js..

In addition, operating system concepts such as multi-threading, inter-process communication (IPC), and file permission management are integrated to enhance reliability and security. This architecture lays the foundation for the algorithms and approaches discussed below.

2. Data Models and Internal Structures

2.1. File Hierarchy Representation

A core part of any file management system is maintaining a structured representation of files and directories. Our approach utilizes a tree-based data structure to model the hierarchy:

Nodes: Represent files or folders; each node contains metadata such as name, size, type, timestamps, and access control settings.

Edges: Define the parent-child relationship, crucial for operations like navigation and inheritance of permissions.

Algorithm Outline (File Creation in Hierarchy):

Input: File or folder name, its path, the user’s credentials, and initial metadata.

Process:

```
    Traverse the tree along the specified path.
    Validate that the user has the necessary permissions at each level.
    Insert the new node at the correct position.
```

Output: Updated tree structure with the new node.

This tree structure enables quick traversal and supports efficient recursive operations for deletion, searching, and permission inheritance.

3. File Operations

File operations—such as creation, deletion, modification, and upload—are at the heart of the system. Each operation has been designed with concurrency, data integrity, and security in mind.

3.1. File Creation and Modification

File Creation:

1. Approach:

   Validate the target directory's path.

   Check for name conflicts.

   Allocate metadata and initialize the file's content container.

   Update the file system tree and the corresponding entry in MongoDB.

2. Pseudocode:

javascript

```
FUNCTION createFile(filePath, fileData, userID)
    Step 1: Validate the file path and retrieve the parent directory node
    parentNode ← getDirectoryNode(filePath)
     Step 2: Check if the user has write permission on the parent
directory
    IF NOT hasPermission(userID, parentNode, "write") THEN
        RAISE ERROR "Access denied"
    END IF
     Step 3: Check for file name conflicts within the parent directory
    IF fileData.name EXISTS IN parentNode.children THEN
        RAISE ERROR "File already exists"
    END IF
    Step 4: Create a new file node with the required metadata
    newNode ← {
        name: fileData.name,
        type: "file",
        content: fileData.content,
        createdAt: CURRENT_DATE_TIME,
        permissions: defaultPermissions(userID)
    }
    Step 5: Update the file hierarchy by adding the new node to the parent's children
    ADD newNode TO parentNode.children
    Step 6: Update the persistent database with the new file node
    updateDatabase(newNode)
    Step 7: Return the newly created file node
    RETURN newNodeEND FUNCTION
```

3.2. File Modification:

1. Approach:

   Implement a 'read-modify-write' cycle with file locking to prevent race conditions.

   Use version control (timestamping or a version number) to manage multiple updates.

2. Key Considerations:

   File locks (either optimistic or pessimistic) are utilized.

   Real-time updates are managed via asynchronous call-backs to update the UI progressively.

3.3. File Deletion

1. Approach:

   Validate deletion rights based on user permissions.

   Traverse the file system tree to locate the node.

   Remove the node and free associated resources in both the file system and MongoDB.

2. Pseudocode:

```
javascript
FUNCTION deleteFile(filePath, userID)
    Step 1: Retrieve the file node using the given file path
    node ← getNode(filePath)


    Step 2: Check if the user has 'delete' permission on the node
    IF NOT hasPermission(userID, node, "delete") THEN
```

```
        RAISE ERROR "Access denied"
    END IF
    **Step 3:** Get the parent directory node of the file
    parentPath ← getParentPath(filePath)
    parentNode ← getDirectoryNode(parentPath)
     **Step 4:** Remove the node from the parent directory's children list
    REMOVE node FROM parentNode.children
    **Step 5:** Delete the file node from the database
    removeFromDatabase(node.id)


     **Step 6:** Return confirmation of successful deletion
    RETURN true
END FUNCTION
```

File Upload and Integrity Verification

File Upload:

1. Approach:

   Implement chunked file uploads for handling large files efficiently.

   Each file is divided into manageable segments, which are uploaded concurrently.

   On the server side, the segments are reassembled, and a checksum (e.g., MD5 or SHA-256) is generated to verify integrity.

2. Workflow:

   Divide file into chunks.

   For each chunk:

      Validate user permissions.

      Upload concurrently using asynchronous operations.

   Reassemble the chunks.

   Verify file integrity post-upload.

3. Algorithm Considerations:

   Use promises or async/await in Node.js to handle multi-threaded upload sessions.

   Store temporary upload states in the database to allow resuming interrupted uploads.

References

[1] Hennessy and Patterson, Computer Architecture: a Quantitative approach, 6th ed. Morgan Kaufmann, 2019.

Available: https://archive.org/details/computerarchitectureaquantitativeapproach6thedition

[2] Ghemawat, Gobioff, and Leung, The Google File System. 19th ACM Symp.

Available: https://pdos.csail.mit.edu/6.824/papers/gfs.pdf

[3] Dean and Ghemawat, MapReduce: Simplified Data Processing on Large Clusters.

Available:https://research.google/pubs/mapreduce-simplified-data-processing-on-large-clusters/

[4] A. S. Tanenbaum and H. Bos, Modern Operating Systems, 4th ed., Pearson, 2015.

Available: https://archive.org/details/modernoperatings0000tane

[5] W. Stallings, Cryptography and Network Security: Principles and Practice, 7th ed., Pearson, 2016.

Available:https://www.pearson.com/en-us/subject-catalog/p/cryptography-and-network-security-principles-and-practice/P200000003477/9780135764213