B.M.S COLLEGE OF ENGINEERING BENGALURU
Autonomous Institute, Affiliated to VTU

OOMD Mini Project Report

Cloud-Based File Storage & Sharing System

Submitted in partial fulfillment for the award of degree of

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by:

Chitrashree K (1BM23CS081)
Dyuthi (1BM23CS097)
Harshitha H G (1BM23CS108)
Hitha Harish (1BM23CS115)

Faculty In charge

Dr. Pallavi G B

Professor

Department of Computer Science and Engineering
B.M.S College of Engineering
Bull Temple Road, Basavanagudi, Bangalore 560 019
2022-2023

B.M.S COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, Chitrashree K (1BM23CS081) , Dyuthi (1BM23CS097), Harshitha H G (1BM23CS108) and Hitha Harish (1BM23CS115) students of 5th Semester, B.E, Department of Computer Science and Engineering, BMS College of Engineering, Bangalore, hereby declare that, this OOMD Mini Project entitled "Cloud-Based File Storage & Sharing System" has been carried out in Department of CSE, BMS College of Engineering, Bangalore during the academic semester August - January 2026. I also declare that to the best of our knowledge and belief, the OOMD mini Project report is not from part of any other report by any other students.

Signature of the Candidate

Chitrashree K (1BM23CS081)
Dyuthi (1BM23CS097)
Harshitha H G (1BM23CS108)
Hitha Harish (1BM23CS115)

BMS COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the OOMD Mini Project titled "**Cloud-Based File Storage & Sharing System"** has been carried out by Chitrashree K (1BM23CS081), Dyuthi (1BM23CS097), Harshitha H G (1BM23CS108) and Hitha Harish (1BM23CS115) during the academic year 2025-2026.

Signature of the Faculty in Charge

Dr. Pallavi G B

Professor

Table of Contents

Chapter 1. Problem Statement

In today's digital era, data has become one of the most valuable assets for individuals, professionals, and organizations. With the increasing dependence on technology and remote connectivity, the need for reliable, secure, and easily accessible storage platforms has grown significantly. Traditional methods of storing data—such as hard drives, pen drives, and local system storage—are often prone to hardware failure, data loss, theft, and limited accessibility. These limitations highlight the importance of cloud-based solutions that offer not only storage but also mobility, scalability, and enhanced security.

The **Cloud-Based File Storage & Sharing System** aims to address these challenges by providing a centralized, cloud-integrated platform for managing digital files. This system enables users to upload, store, organize, and share their files from any device, at any location, and at any time. With the advancement of cloud technologies and distributed computing, such systems have become essential for improving productivity, enabling collaboration, and ensuring continuous data availability.

Unlike traditional storage platforms, cloud-based solutions offer several benefits such as real-time synchronization, automated backups, disaster recovery, and cross-platform accessibility. Users can share files through secure links or role-based permission settings, ensuring that data is shared only with authorized individuals. The integration of modern security standards such as **AES-256 encryption**, **TLS 1.3**, and **JWT/OAuth 2.0 authentication** ensures that user data remains protected from unauthorized access, cyber threats, and tampering.

Furthermore, the backend of the system leverages robust and scalable cloud storage services like **AWS S3** or **Google Cloud Storage**, which are widely recognized for their reliability, durability, and high performance. These services enable seamless handling of large volumes of data while maintaining low latency and consistent performance. By using RESTful APIs, the system ensures smooth communication between the frontend client, backend server, and cloud infrastructure.

From an educational and technical perspective, this project demonstrates the application of key **Object-Oriented Programming (OOP)** concepts such as abstraction, encapsulation, modularity, and reusability. It also reflects modern software engineering practices, including secure authentication mechanisms, cloud integration, microservice-oriented backend architecture, and responsive user interfaces.

Overall, the **Cloud-Based File Storage & Sharing System** serves as an innovative and practical solution to modern file management challenges. It not only enhances data accessibility and collaboration but also ensures strong protection of user information through cutting-edge technologies. This project showcases the importance of cloud computing in delivering reliable, scalable, and secure digital solutions for personal, academic, and organizational use.

Motivation

The motivation behind developing the Cloud-Based File Storage & Sharing System stems from the growing need for secure, reliable, and easily accessible digital storage solutions in an increasingly connected world. As the volume of digital data continues to rise at an exponential rate, individuals and organizations face significant challenges in managing, protecting, and sharing their files efficiently. Traditional storage devices such as USB drives, external hard disks, and local system storage, while convenient, come with several drawbacks—limited capacity, vulnerability to physical damage, risk of data loss, and restricted accessibility. These limitations motivated the creation of a cloud-integrated platform that overcomes these issues while offering enhanced functionality and user convenience.

Another major motivating factor is the shift toward remote work, online learning, and digital collaboration. In today's world, users require instant access to important files regardless of their location or device. Industries, educational institutions, and businesses depend heavily on seamless file sharing and collaboration to maintain productivity. A cloud-based solution ensures real-time synchronization, multi-device compatibility, and convenient sharing features, enabling users to work together efficiently without geographical constraints.

Security concerns also play a crucial role in motivating the development of this system. Cyber threats, unauthorized data access, and privacy breaches have become common, making it essential to protect sensitive information using modern security mechanisms. Implementing technologies such as encryption, secure authentication, and controlled permissions ensures that user data remains confidential and protected at all times. This system addresses these critical concerns by integrating robust security protocols that safeguard data from both internal and external threats.

Additionally, the motivation arises from the desire to build a platform that is scalable and future-ready. Cloud infrastructure provides the capability to handle large volumes of data without compromising performance. Unlike traditional systems that require manual upgrades, cloud-based systems can dynamically scale according to user needs. This scalability makes the system ideal for both personal and organizational use.

From a learning and development perspective, this project is motivated by the opportunity to apply Object-Oriented Programming principles and real-world software engineering practices. It offers practical experience in working with cloud technologies, RESTful APIs, secure authentication systems, and dynamic user interfaces. Developing such a solution not only enhances technical skills but also provides valuable insights into building enterprise-level applications.

In summary, the motivation for creating the **Cloud-Based File Storage & Sharing System** is driven by the need for a modern, secure, and efficient file management solution that addresses the limitations of traditional storage methods, supports remote collaboration, ensures high-level data security, and leverages the power of cloud computing. This project aims to provide a reliable and future-oriented platform that meets the evolving demands of today's digital ecosystem.

Chapter 2: Software Requirement Specification

Cloud-Based File Storage & Sharing System

1. Introduction:

1.1 Purpose

The purpose of this document is to define the software requirements for the Cloud-Based File Storage & Sharing System. This SRS outlines the functional and non-functional specifications of the system, providing a clear understanding of its goals, architecture, and design constraints. The document serves as a guideline for developers, project managers, testers, and stakeholders involved in the project.

1.2 Document Conventions

This document follows the IEEE Standard for Software Requirements Specifications (IEEE 830). The terms "shall" indicate mandatory requirements, while "should" indicates optional or recommended features. All diagrams are represented using Mermaid UML syntax for clarity and portability.

1.3 Intended Audience and Reading Suggestions

This SRS is intended for:
Developers – To understand detailed functional requirements and design constraints.
Project Managers – To plan, schedule, and track the project development.
Testers – To design and perform validation and verification procedures.
Clients/End Users – To understand the capabilities and limitations of the system. It is suggested that readers review Sections 1 and 2 for an overview, followed by Section 3 for detailed system functionality.

1.4 Product Scope

The Cloud-Based File Storage & Sharing System is designed to allow users to securely store, access, and share files via a web-based and mobile platform. It enables functionalities such as file upload/download, sharing via links or user permissions, version control, and real-time synchronization. The system aims to enhance collaboration and data availability while maintaining data security and integrity.

1.5 References

IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications.
OWASP Security Guidelines for Web Applications.
Amazon Web Services (AWS) Cloud Storage Documentation.
Google Cloud Storage API Reference.

2. Overall Description:

2.1 Product Perspective

The Cloud-Based File Storage & Sharing System operates as a distributed, client-server architecture hosted on a cloud infrastructure (e.g., AWS, Azure, or GCP). The system will include both a web interface and mobile application, communicating with backend APIs for authentication, data management, and file storage.

2.2 Product Features

User Authentication and Authorization: Secure login, registration, and role-based access control.
File Management: Upload, download, rename, delete, and organize files and folders.
Sharing and Collaboration: Share files with specific users or generate public/private shareable links.
Version Control: Maintain file history and revert to previous versions.
Search and Filtering: Locate files quickly using search keywords, filters, or tags.
Storage Analytics: Display usage statistics and storage capacity reports.
Backup and Recovery: Automated cloud backups with recovery options.

2.3 User Classes and Characteristics

End User – Uploads, downloads, and shares files using a simple, user-friendly interface.
Administrator – Manages users, permissions, and overall storage operations.
System Developer – Develops and maintains the system's backend and frontend components.
Tester – Validates system functionality and ensures quality through testing.

2.4 Operating Environment

Server: Cloud-hosted environment (AWS EC2, Azure VM, or GCP Compute Engine)
Client: Web browsers (Chrome, Firefox, Edge, Safari), Android/iOS apps
Database: MySQL or PostgreSQL for metadata; Cloud object storage for files
Operating Systems: Windows, Linux, macOS

2.5 Design and Implementation Constraints

Must comply with cloud provider APIs (e.g., AWS S3 or Google Cloud Storage).
Implementation languages: Java/Spring Boot for backend, React/Flutter for frontend.
Follow REST API standards for interoperability.
Use encryption standards (AES-256 for data, TLS 1.3 for communication).

2.6 User Documentation

Online help section integrated into the app.
User manual with setup and troubleshooting guides
Administrator guide for managing user accounts and permissions.

2.7 Assumptions and Dependencies

Users must have stable internet connectivity.
The system depends on third-party cloud service uptime (e.g., AWS or Google Cloud).

Browser and device compatibility as per supported versions.

3. System Features

3.1 Functional Requirements

User Registration & Login – Allows users to register and securely authenticate using email, password, and token-based authentication (JWT/OAuth 2.0).

File Upload and Storage – Enables users to upload files to cloud storage while saving related metadata like name, size, and owner.

File Sharing – Allows users to share files through permissions or shareable links, with options to revoke access anytime.

File Download and Access Control – Permits only authorized users to download files, maintaining detailed audit logs of al download events.

Version Control and History – Maintains file version history and allows users to restore previous versions when needed.

Search and Organization – Supports file search by name, type, or tags and provides folder-based hierarchical organization.

3.2 Non-Functional Requirements

Security: Data encryption at rest and in transit.

Scalability: Support at least 10,000 concurrent users.

Performance: Response time under 2 seconds for all major operations.

Availability: 99.9% uptime SLA.

Reliability: Automatic recovery from failures within 5 minutes.

Usability: User-friendly UI with consistent design across devices.

Maintainability: Modular code structure with proper documentation.

Portability: Compatible with major browsers and OS.

3.3 Domain Requirements

Compliance with data protection laws (e.g., GDPR, HIPAA).

Support for role-based access control (Admin, User, Viewer).

All files stored with unique identifiers for deduplication.

3.4 External Interface Requirements

User Interface: Responsive web dashboard and mobile app.

Hardware Interface: Standard input devices and mobile touchscreens.

Software Interface: Integration with Google Drive, Dropbox APIs.

Communication Interface: REST API over HTTPS.

4. Appendix

4.1 Acronyms and Abbreviations

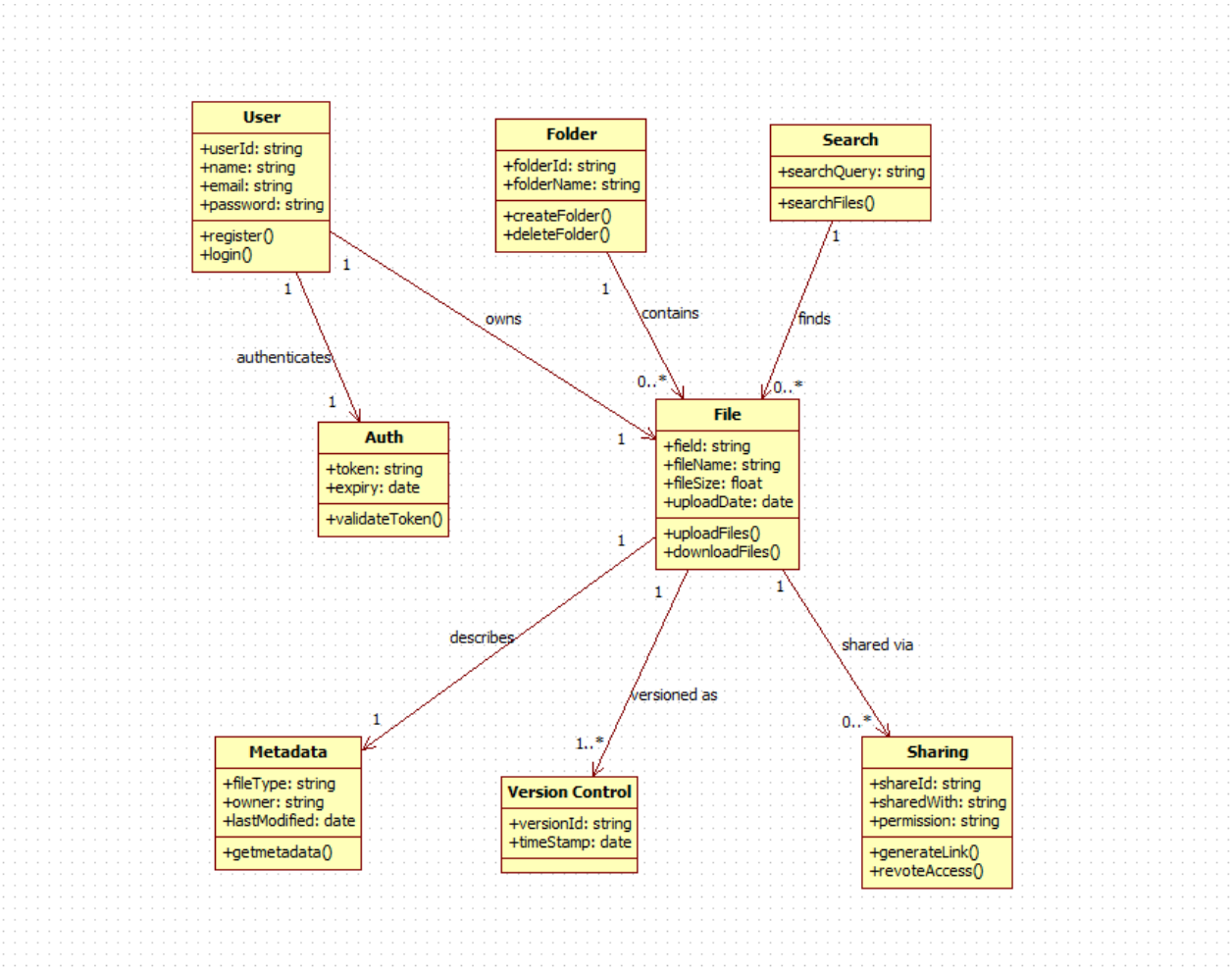
Acronym	Definition
API	Application Programming Interface
AWS	Amazon Web Services
GDPR	General Data Protection Regulation

JWT JSON Web Token
UI User Interface
REST Representational State Transfer

4.2 Glossary

Cloud Storage: A model of data storage where digital data is stored on remote servers.
Access Control: Restriction of access to system resources to authorized users.
Encryption: The process of converting information into code to prevent unauthorized access.
Backup: A copy of data stored separately for recovery in case of loss.
Scalability: The capability of a system to handle a growing amount of work by adding resources.

Chapter 3: Class Modelling



Class Diagram

Explanation:

The class diagram represents the structural blueprint of the **Cloud-Based File Storage & Sharing System**, illustrating the main system components (classes), their attributes, methods, and the relationships between them. Each class models a key part of the system's functionality, following Object-Oriented Programming principles such as abstraction, modularity, and encapsulation.

a) User Class

The **User** class represents individuals who interact with the system.

Attributes:

- userId: Unique identifier for each user
- name: Name of the user
- email: Registered email address
- password: Encrypted user password

Methods:

- register(): Allows new users to create an account
- login(): Authenticates existing users

Relation:

A **User** is authenticated through an **Auth** object (1-to-1 association). A **User** also **owns** one or more **Folders** (1-to-many).

b) Auth Class

The **Auth** class handles authentication and token management.

Attributes:

- token: Session or access token generated during login
- expiry: Token expiration time

Methods:

- validateToken(): Verifies token integrity and expiration

Relation:

Each User has exactly one active Auth token at a time (1–1).

c) Folder Class

The Folder class models the directory structure used to organize files.

Attributes:

- folderId: Unique identifier
- folderName: Name assigned by the user

Methods:

- createFolder(): Generates a new folder
- deleteFolder(): Removes an existing folder

Relation:

A **User** owns multiple **Folders** (1-to-many).

A **Folder** contains multiple **Files** (1-to-many).

d) File Class

The File class represents the core entity stored and managed in the cloud.

Attributes:

- fileId: Unique file identifier
- fileName: Name of the file
- fileSize: Size of the file in MB/KB
- uploadDate: Timestamp when the file was uploaded

Methods:

- uploadFiles(): Handles uploading of files
- downloadFiles(): Handles downloading

Relation:

- A **Folder** contains 0..* **Files**
- A **File** may have **multiple versions** via the Version Control class
- A **File** has exactly **one Metadata** description
- A **File** can be **shared** through the Sharing class
- The Search class finds multiple files (Search → File is 1 to 0..*)

This makes **File** the central entity of the diagram.

e) Metadata Class

The Metadata class provides descriptive information about a file.

Attributes:

- fileType: Type/format of the file (pdf, jpg, mp4, etc.)
- owner: File owner
- lastModified: Timestamp of the latest update

Methods:

- getMetadata(): Returns metadata details

Relation:

Each File has exactly one Metadata object (1–1).

f) Version Control Class

This class handles file versioning.

Attributes:

- versionId: Unique version identifier
- timeStamp: Time when this version was created

Relation:

A File is versioned as 1-to-many (one file can have multiple versions).

g) Sharing Class

The **Sharing** class manages sharing permissions.

Attributes:

- shareId: Unique share identifier
- sharedWith: User or group the file is shared with
- permission: Level of access (view, edit, download, etc.)

Methods:

- generateLink(): Creates a sharable link
- revokeAccess(): Removes access permissions

Relation:

A **File** can be shared in multiple ways (File → Sharing is 1-to-many).

h) Search Class

The **Search** class provides search functionality.

Attributes:

- searchQuery: User-entered query string

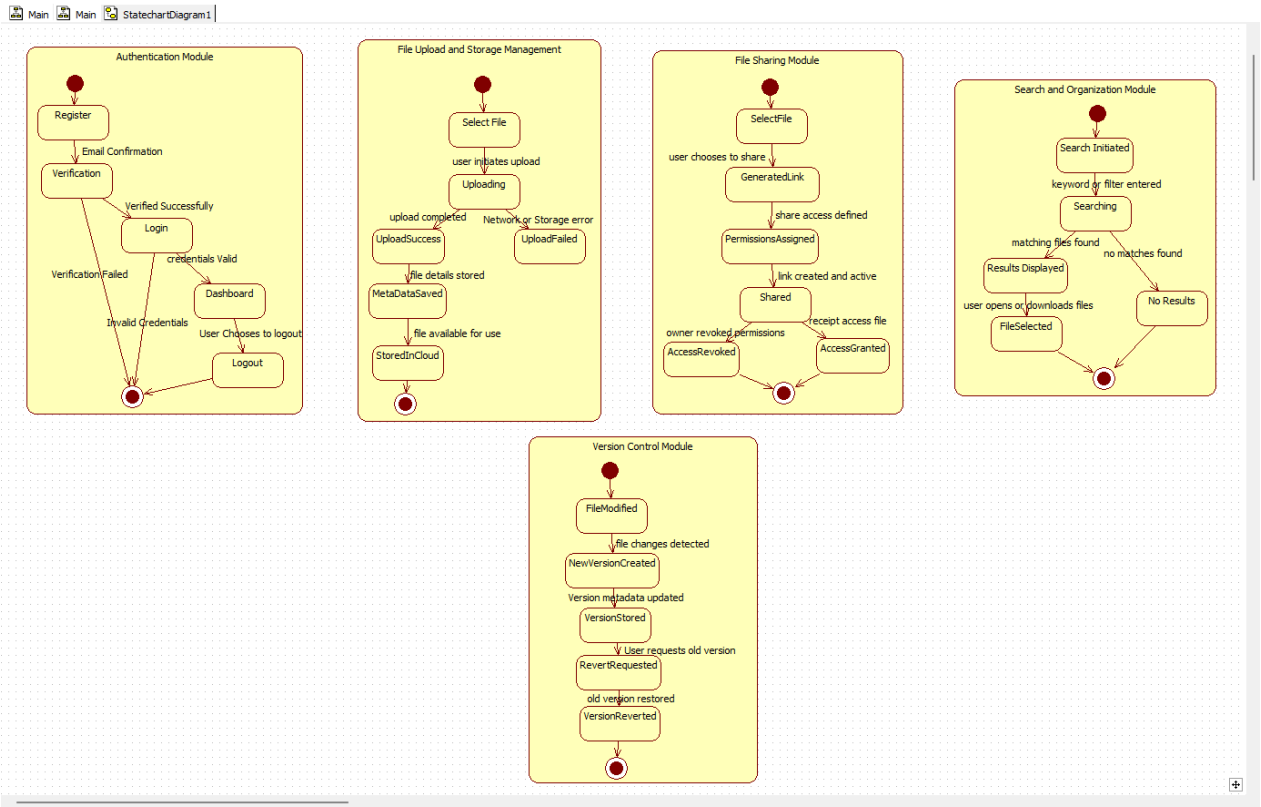
Methods:

- searchFiles(): Finds files that match the query

Relation:

Search can find multiple files (1-to-many).

Chapter 4: State Modelling



Statechart Diagram

The Statechart Diagram represents the dynamic behavior of the Cloud-Based File Storage & Sharing System, illustrating how the system transitions between different states in response to user actions and system events. It is divided into five major modules, each showing the lifecycle of key operations such as authentication, file upload, sharing, searching, and version control. This model highlights event-driven state changes and ensures clarity in understanding how the system behaves under different scenarios.

a) Authentication Module:

This module shows the complete sequence of states involved in user registration, verification, login, and logout.

State Transitions:

- Initial State → Register
  The process begins when the user initiates registration.
- Register → Email Confirmation → Verification
  After providing details, an email verification step is triggered.
- Verification → Verified Successfully → Login → Dashboard
  Once the user is verified, they can log in and access the dashboard.
- Verification → Verification Failed → Final State
  If details are incorrect or verification fails, the process ends.
- Login → Invalid Credentials → Final State

Incorrect login attempts lead to termination of the process.

- Dashboard → Logout → Final State
  A logged-in user may choose to logout, ending the session.

b) File Upload & Storage Management Module

This module models how files are selected, uploaded, stored, and made available in the cloud.

State Transitions:

- **initial State → Select File → Uploading**
  The process begins when the user selects a file and initiates upload.
- **Uploading → Upload Success → Metadata Saved → Stored in Cloud → Final State**
  On successful upload, metadata is saved, and the file becomes available for use.
- **Uploading → Upload Failed → Final State**
  Errors due to network issues or storage failure end the process abruptly.

c) File Sharing Module:

This module represents the lifecycle of sharing a file with other users.

State Transitions:

- **Initial State → Select File → Generated Link**
  The user chooses a file and the system generates a sharable link.
- **Generated Link → Permissions Assigned → Shared**
  Access controls such as read/edit permissions are applied before the file becomes shared.
- **Shared → Access Granted → Final State**
  The receiver obtains file access successfully.
- **Shared → Access Revoked → Final State**
  The owner withdraws permissions, terminating sharing access.

This module ensures controlled and revocable sharing functionality.

d) Search & Organization Module

This module shows how users find and access stored files.

State Transitions:

- **Initial State → Search Initiated → Searching**
  The user enters a keyword or filter to begin the search.
- **Searching → Results Displayed → File Selected → Final State**
  If matches are found, results appear and the user can open or download selected files.
- **Searching → No Results → Final State**
  If no matching files exist, the process ends.

This module depicts the simplicity and responsiveness of the system's search feature.

e) Version Control Module

This module represents file versioning operations when modifications are made.

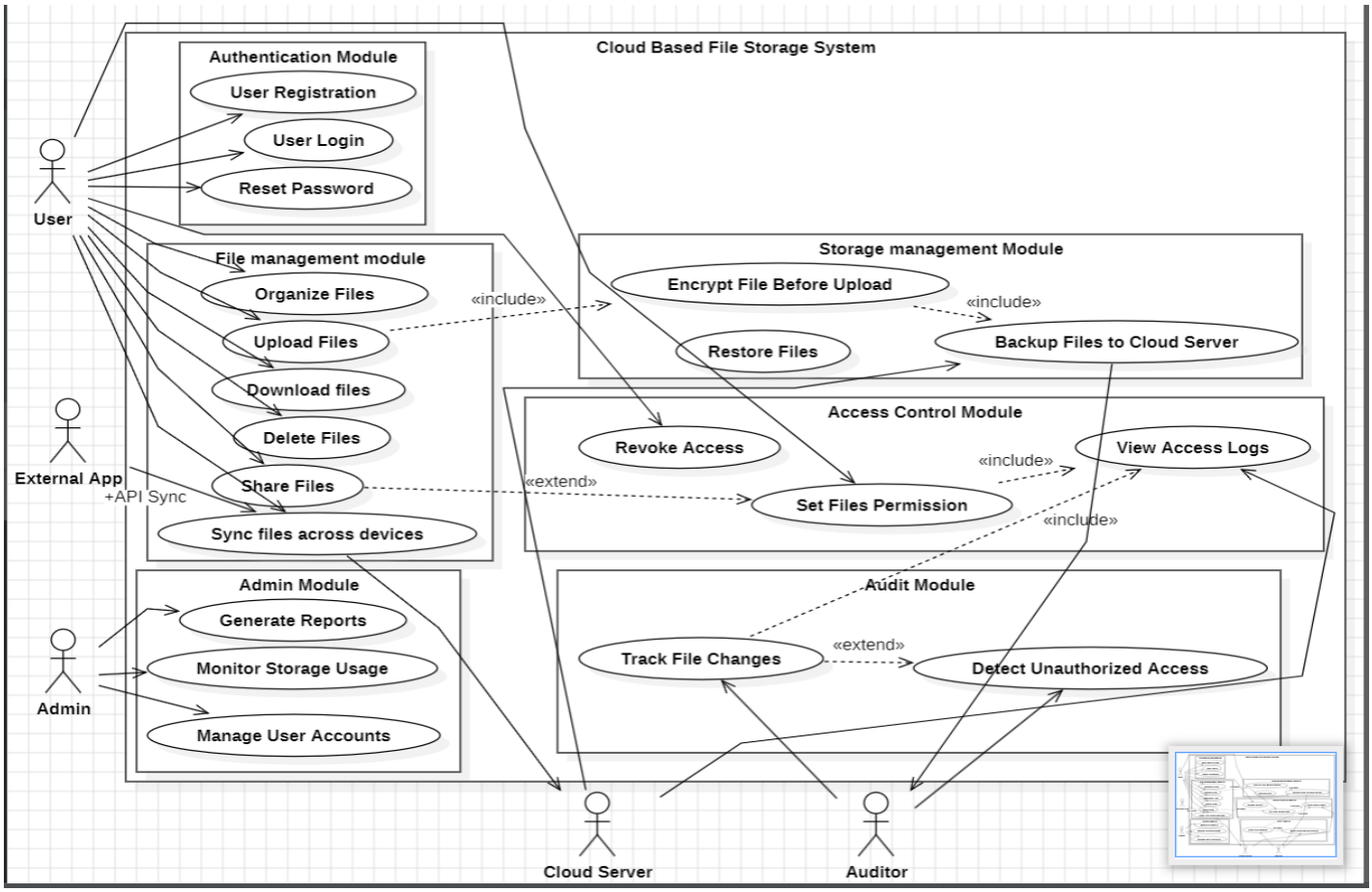
State Transitions:

- **Initial State → File Modified → New Version Created**
  When a file is edited, the system detects changes and generates a new version.
- **New Version Created → Version Stored → Final State**
  Updated metadata and version history are saved.
- **Version Stored → Revert Requested → Version Reverted → Final State**
  If a user requests an older version, the system restores it.

This module ensures integrity, traceability, and recoverability of files.

Chapter 5: Interaction Modeling:

I. Use Case Diagram:



Your use case diagram provides a comprehensive overview of how different actors interact with the Cloud-Based File Storage System. It organizes the system into multiple functional modules and shows how these modules work together to support file storage, access, security, administration, synchronization, and auditing.

A) Actors in the System

i. User

A regular user who interacts with the system for file-related operations.

They can:

- Register/login
- Upload/download/delete files
- Share files
- Sync files across devices
- Organize files

ii. **External App**

Represents third-party services (e.g., mobile apps, external integrations) that sync files with the system through APIs.

iii. **Admin**

The administrator is responsible for system-level management tasks:

- Manage user accounts
- Generate system reports
- Monitor storage usage

iv. **Cloud Server**

Acts as the storage and computation backend. It is responsible for file storage, backup, encryption processing, and restoring files.

v. **Auditor**

A security monitoring entity responsible for:

- Tracking file changes
- Detecting unauthorized access attempts
- Viewing access logs

B) Major System Modules and Their Use Cases

i. **Authentication Module**

Use Cases

- User Registration
  Allows new users to sign up and create an account.
- User Login
  Authenticates existing users and enables access to the system.
- Reset Password
  Helps users recover or change their password.

Purpose

Ensures authorized access, user identity verification, and secure entry into the system.

ii. **File Management Module**

This is the core module where users manage their files.

Use Cases

- Organize Files
  Create folders, move files, rename items, etc.
- Upload Files
  Users send files from their device to the cloud.
- Download Files

Retrieve files from the cloud to local devices.

- Delete Files
  Remove files permanently or move to recycle bin (depending on implementation).
- Share Files
  Users can send links or allow permissions to other users.
- Sync Files Across Devices
  Ensures files stay consistent across multiple devices.
  External Apps interact here via API Sync.

Relationships

- Upload Files includes "Encrypt File Before Upload" (from the Storage Module).

iii. Storage Management Module

Manages the storage infrastructure and file preparation.

Use Cases

- Encrypt File Before Upload
  Ensures files are encrypted for privacy before they reach the cloud.
- Backup Files to Cloud Server
  Periodically creates backup copies for recovery.
- Restore Files
  Enables users to recover files from backups.

Purpose

Provides data protection, disaster recovery, and encryption support for secure storage.

iv. Access Control Module

Controls permissions and file accessibility.

Use Cases

- **Set File Permissions**
  Assign read/write/view/share permissions.
- **Revoke Access**
  Remove previously granted access to specific users.
- **View Access Logs**
  Helps admins/auditors see who accessed what and when.

Relationships

- "Set File Permission" **includes** "View Access Logs"
- "Share Files" (from File Management) **extends**

"Set File Permission" — because sharing requires granting permissions.

v. **Admin Module**

This module is exclusively for administrators.

Use Cases

- **Generate Reports**
  Storage usage reports, activity summaries, system health reports.
- **Monitor Storage Usage**
  Tracks how much space users are consuming.
- **Manage User Accounts**
  Add, remove, suspend, or update user profiles.

Purpose

Supports system-wide maintenance and administrative oversight.

vi. **Audit Module**

Ensures security monitoring and compliance.

Use Cases

- **Track File Changes**
  Logs all edits, uploads, deletions, and restorations.
- **Detect Unauthorized Access**
  Alerts when unusual or suspicious activity is detected.
  This use case **extends** "Track File Changes" — because detection depends on change tracking.

Purpose

To maintain **security transparency**, detect intrusions, and assist in forensic analysis.

C) Important UML Relationships Present

Include (→)

Means one use case *always* requires another:

- Uploading files includes Encrypting before upload
- Setting file permissions includes viewing logs
- Backup process includes storage operations

Extend (-->)

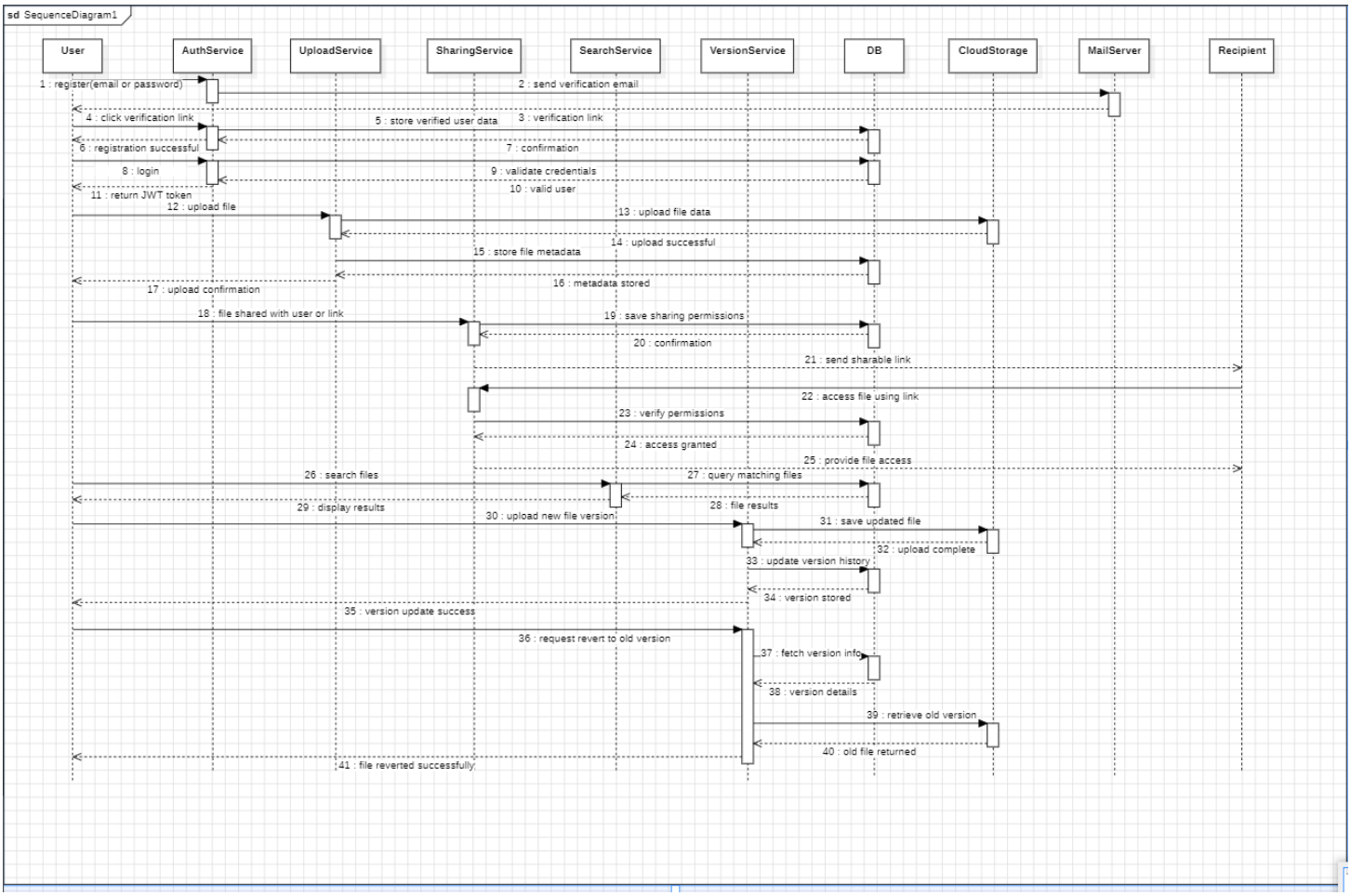
Means a use case optionally adds behavior:

- Sharing files extends setting permissions
- Detect unauthorized access extends tracking changes

Actors interact → modules → cloud server

Shows how each actor connects at different levels of the system.

II. Sequence Diagram

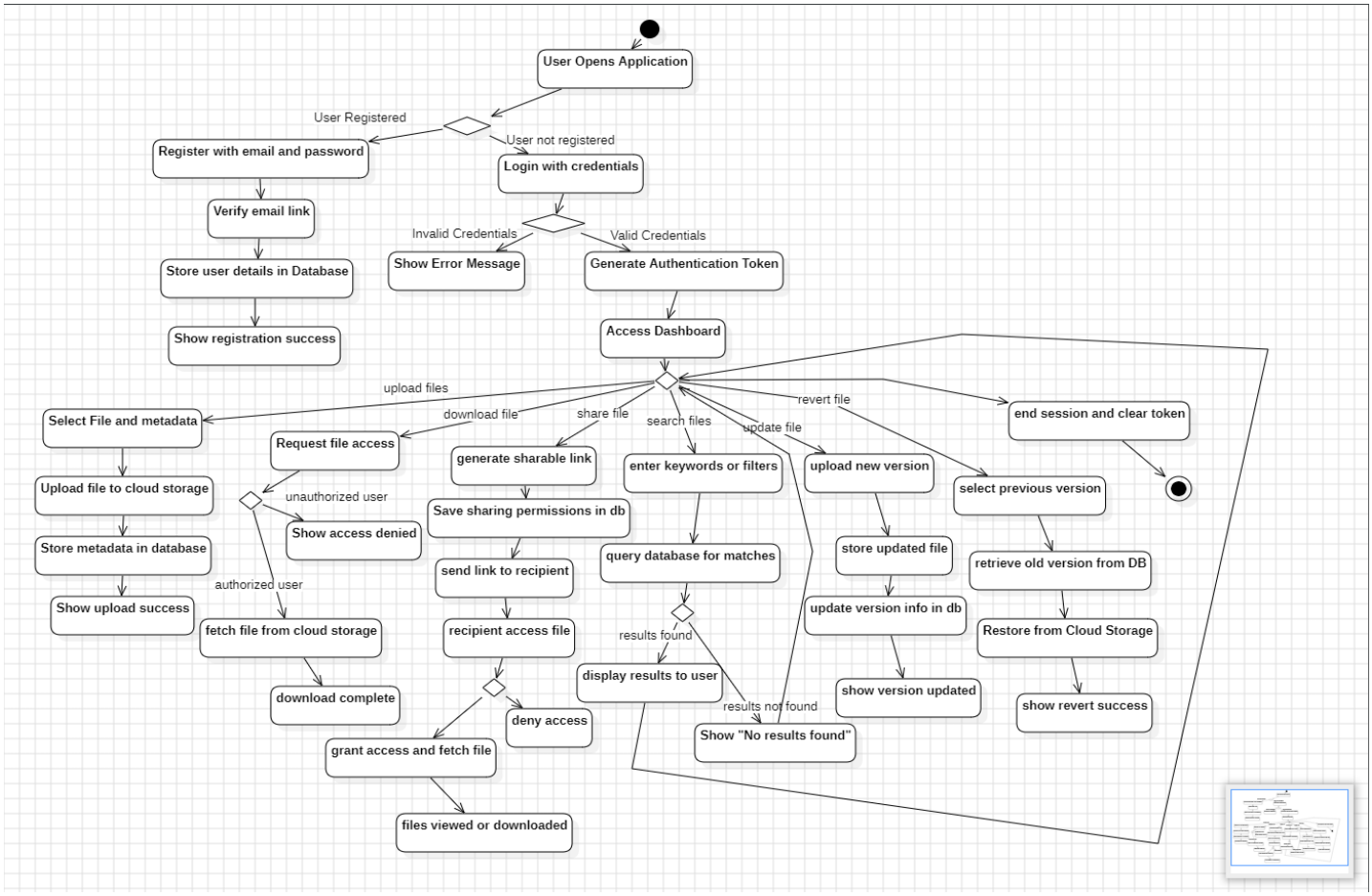


The sequence diagram illustrates the complete workflow of the Cloud-Based File Storage System, showing how different services interact from the moment a user registers until file versioning and restoration are performed. The process begins when a user submits registration details, triggering the AuthService to send a verification email through the MailServer. After receiving the verification link, the user clicks it, and AuthService validates the account, stores the verified user information in the database, and confirms successful registration. The user then logs in, and AuthService verifies credentials before issuing a JWT token, enabling secure authenticated access to the system. Once authenticated, the user uploads a file through the UploadService, which sends the file to CloudStorage and stores the corresponding metadata in the database. After the upload succeeds, the user can share the file using the SharingService, which saves sharing permissions in the database and sends a shareable link to the intended recipient. When the recipient accesses the link, the system verifies permissions and retrieves the file from CloudStorage, ensuring secure controlled access.

The diagram also captures the search functionality, where the user initiates a query, the

SearchService retrieves matching files from the database, and the results are returned to the user. Additionally, the version control flow is shown: when the user uploads a modified file, the UploadService saves the updated file in CloudStorage, while the VersionService updates the version history in the database and confirms the version creation. If the user requests to revert to an older version, VersionService fetches historical version details from the database and retrieves the corresponding file from CloudStorage, ultimately restoring it for the user. Throughout the entire sequence, each service interacts seamlessly with shared components like the database and CloudStorage, reflecting a fully integrated cloud file management ecosystem.

III. Activity Diagram



The activity diagram illustrates the complete workflow of a Cloud-Based File Storage System, starting from the moment the user opens the application. When the application launches, the system checks whether the user is already registered; if not, the user proceeds through registration by providing an email and password, verifying the email link, and finally having their details stored in the database, after which a registration success message is shown. If the user is already registered, they instead log in with their credentials. Invalid login attempts display an error message, while valid credentials trigger the generation of an authentication token, granting access to the main dashboard. From the dashboard, the user can perform various actions such as uploading, downloading, sharing, searching, updating, or reverting files.

When uploading files, the user selects a file and its metadata, uploads it to cloud storage, and the metadata is stored in the database before showing an upload success message. For file download, access is requested and verified; unauthorized users see an access-denied message, while authorized users retrieve the file from cloud storage until the download completes. Sharing a file begins by generating a sharable link and saving permissions to the database. When the recipient accesses the link, the system checks their permissions—access is either denied or granted. If granted, the authorized recipient can fetch, view, or download the file.

Searching files begins when the user enters keywords or filters, prompting the system to query the

database. If matches are found, results are displayed; otherwise, a “No results found” message is shown. Updating a file version involves uploading a new version, storing the updated file, updating version information in the database, and notifying the user that the version has been updated. Reverting a file requires selecting a previous version, retrieving the old version from the database, restoring it from cloud storage, and showing a success message. Finally, the user may choose to end the session at any time, which clears the authentication token and completes the activity flow.

Chapter 6: UI Design

1. Auth Page

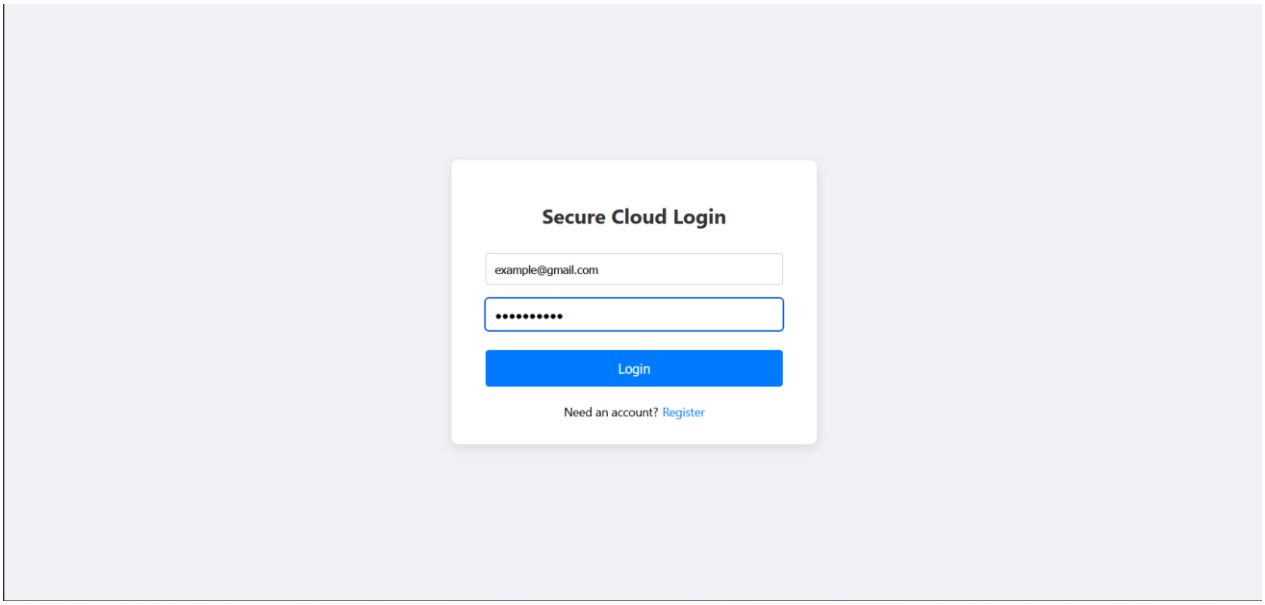


Fig 1. Login Page

This screen represents the implementation of the **Authentication Module**, which is the critical starting point defined in the project's **Software Requirements Specification (SRS)** and detailed in the **Activity** and **State Diagrams**. The user interface (UI) features a clean, centrally focused **Login** card, ensuring adherence to the **Usability** non-functional requirement. It includes standard fields for **Email** and **Password** to capture user credentials for verification. The inclusion of the "Need an account? **Register**" link provides immediate access to the registration flow, directly supporting the **User Registration & Login** functional requirement. Upon successful submission (validated credentials), the system transitions the user from the Login state to the Dashboard state, allowing access to the core **File Management** functionality, as illustrated in the State Diagram.

2. Dashboard

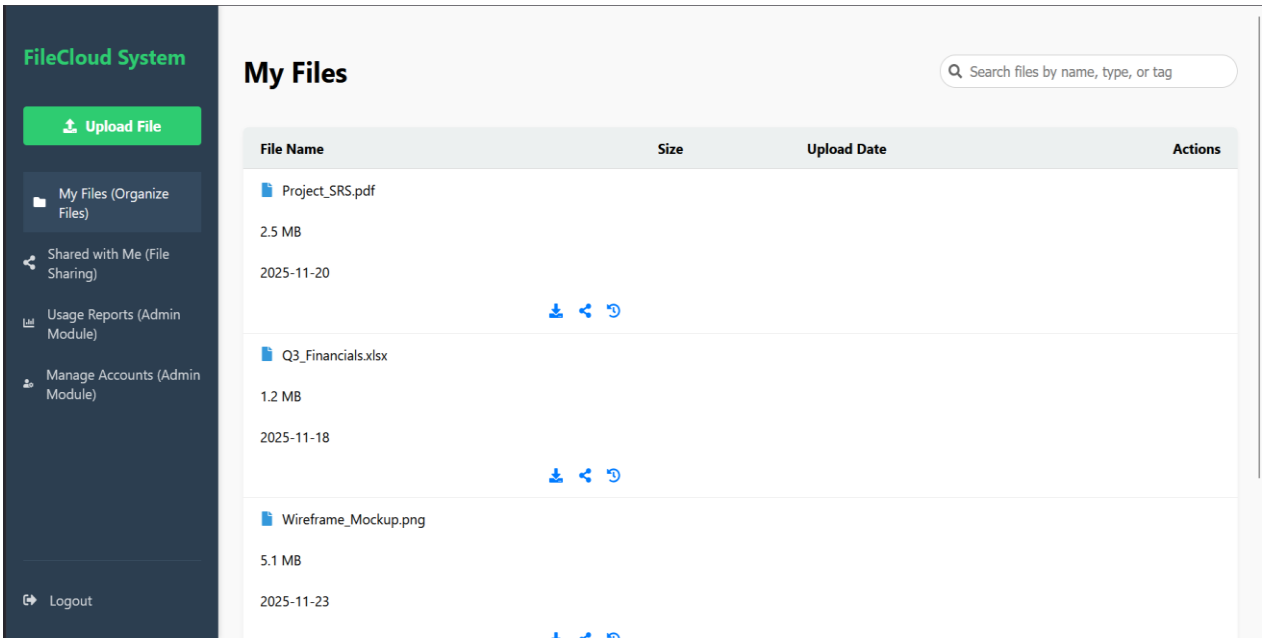


Fig 2. Dashboard Page

This screenshot displays the primary user interface of the system, the **File Management Dashboard**, which is accessed immediately after successful user authentication, fulfilling the core requirements of the **File Management Module** and the **Search and Organization Module**. The layout adheres to standard web usability guidelines with two main sections:

1. **Sidebar:** The left navigation panel provides access to organizational features (**My Files**, **Shared with Me**) and administrative functions (**Usage Reports**, **Manage Accounts**), as defined in the **Use Case Diagram** . The prominent green **"Upload File"** button directly supports the file upload process detailed in the **Activity Diagram**.
2. **Main Content:** This area lists files with attributes like **File Name**, **Size**, and **Upload Date**, derived from the **File** and **Metadata** entities in the **Class Diagram**. The integrated search bar satisfies the **Search and Organization Module** requirement, allowing files to be quickly located. Crucially, the **Actions** column includes icons for **Download**, **Share**, and **Version History**, visually representing the successful implementation of the **File Download and Access Control**, **File Sharing**, and **Version Control** functional requirements.