A PROJECT REPORT

on

STORAGE & FILE SHARING SYSTEM

Submitted by

Ms. Samdisha Ratanlal Vishwakarma

in partial fulfillment for the award of the degree
of

BACHELOR OF SCIENCE in **COMPUTER SCIENCE**

under the guidance of

Prof. Jisha Marathe

Department of Computer Science

MARWARI VIDYALAYA SANCHALIT SMT. KAMALADEVI GAURIDUTT MITTAL COLLEGE OF COMMERCE & SCIENCE

Accredited By Naac: B+ (Iso 9001:2008 Certified)

(2024-2025)

SMT. KAMALADEVI GAURIDUTT MITTAL COLLEGE of COMMERCE AND ARTS

(Affiliated to University of Mumbai)

CERTIFICATE

B.Sc. (C.S.) Semester – 6
(2024-2025)

This is to certify Ms.: **Samdisha Ratanlal Vishwakarma**.
Student of **Bachelor of Science in Computer Science**, Semester – 6th.
Seat No.: **3009396** has successfully completed project implementation in partial fulfillment as per the syllabus prescribed by University of Mumbai.

_____ _____

Professor In-charge
Name:
Date:

BSc. (IT/CS Co-Ordinator)
Name:
Date:

_____

External Examiner
Name:
Date:

ABSTRACT

The **Storage and File Sharing** Platform is a web-based application designed to provide users with a seamless and efficient way to upload, organize, manage, and share files. Built using React 19, Next.js 15, Appwrite, TypeScript, TailwindCSS, and ShadCN, the platform ensures a modern, scalable, and user-friendly experience. The system enables users to store various file types, including documents, images, videos, and audio files, making file management more accessible and collaborative.

The platform consists of several key modules, including User Authentication, File Upload and Management, File Sharing, Dashboard, Global Search, and Sorting Options. The User Authentication module, powered by Appwrite, allows users to securely register, log in, and manage their accounts. The File Upload module supports drag-and-drop functionality, ensuring a smooth and efficient way to store files in the cloud. Users can view, rename, delete, and download files, maintaining complete control over their data.

The File Sharing module enables users to generate shareable links, facilitating easy collaboration. The Dashboard module provides insights into total and consumed storage, recent uploads, and categorized file statistics. A Global Search feature allows users to quickly locate files, while the Sorting module enables efficient organization based on name, size, or upload date.

The platform also incorporates responsive design principles, ensuring a consistent experience across desktop and mobile devices. Security is prioritized, with user authentication, role-based access control, and encrypted file storage ensuring data protection. The integration of React 19's concurrent rendering and Next.js 15’s enhanced server-side capabilities optimize performance, enabling fast file operations and real-time updates.

By leveraging modern web technologies, this Storage and File Sharing Platform provides an intuitive, high-performance, and secure solution for personal and professional file management needs.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to all those who have been instrumental in the successful completion of my individual project during my college journey.

First and foremost, I would like to extend my heartfelt thanks to my project advisor, prof. **Jisha Marathe** for their invaluable guidance, mentorship, and unwavering support throughout the entire project. Their expertise, patience, and willingness to share knowledge played a pivotal role in shaping the project's direction and enhancing its quality.

I am also deeply thankful to the faculty members of the Bsc.IT/CS Department at Smt. K.G. Mittal College for providing a conducive learning environment and access to resources that facilitated my research and development process.

I would like to acknowledge the contributions of my classmates and friends who offered valuable insights, feedback, and moral support. Their encouragement and willingness to brainstorm ideas with me greatly enriched the project.

Furthermore, I appreciate the assistance I received from the Department of Computer Science, who provided access to specialized equipment and software crucial for my project's execution.

Last but not least, I extend my gratitude to all the Teachers and My Classmates who took the time to Guide and encourage, as their input was indispensable for gathering valuable data.

In conclusion, this project would not have been possible without support of all those mentioned above. Thank you for being a part of this journey and for helping me achieve my academic goals.

Sincerely,
Samdisha Ratanlal Vishwakarma

Smt. Kamaladevi Gauridutt Mittal College of Arts & Commerce
7th March 2025

DECLARATION

I hereby declare that the project entitled, "**Storage and File Sharing System**" done at **Smt. K.G. Mittal College of Arts and Commerce** has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

The project is done in partial fulfillment of the requirements for the award of degree of **BACHELOR OF SCIENCE IN COMPUTER SCIENCE** to be submitted as VI - semester project as part of our curriculum.

Samdisha Ratanlal Vishwakarma

PERFORMA FOR THE APPROVAL PROJECT PROPOSAL

(Note: All entries of the proforma of approval should be filled up with appropriate and complete information. Incomplete proforma of approval in any respect will be summarily rejected.)

PNR No.: ……..……... Roll No.: ……

1. Name of the student

_____

2. Title of the Project

_____

3. Name of the Guide

_____

4. Teaching experience of the Guide _____

5. Is this your first submission? Yes _____ No _____

Signature of the Student **Signature of the Guide**

Date: ……... Date: …….

Signature of the Coordinator

Date: ……...

TABLE OF CONTENT

CHAPTER NO. 01
INTRODUCTION

1.1 App Overview

The **Storage and File Sharing System** is a modern and efficient solution for managing and sharing files seamlessly. Built using the latest Next.js 15 and Appwrite Node SDK, this platform offers a user-friendly experience with advanced features to enhance file management. Whether you need to upload, organize, download, or share files, this platform provides a secure and intuitive environment.

With features like user authentication, global search, sorting options, and a dynamic dashboard, users can effortlessly keep track of their stored data. The responsive UI ensures accessibility across all devices, making file management smoother than ever.

This project follows best practices in code architecture and reusability, making it an excellent learning resource for developers. Whether you're an individual user or a team looking for an efficient file-sharing solution, this platform is designed to meet your needs.

1.2 Objectives

- To develop a **secure and user-friendly** platform for uploading, managing, and sharing files.
- To implement **user authentication** using Appwrite, ensuring data privacy and access control.
- To enable **quick and efficient file uploads, downloads, and sharing** through an intuitive interface.
- To provide a **comprehensive dashboard** that displays storage insights, recent uploads, and file statistics.
- To incorporate **global search and sorting features**, allowing users to find and organize files effortlessly.

- To ensure **cross-device compatibility** with a modern, responsive UI for desktops, tablets, and mobile devices.

1.3 Purpose & Scope

Purpose:

The **Storage and File Sharing System** is designed to provide users with a secure, efficient, and user-friendly platform for uploading, managing, and sharing files. With the increasing need for digital storage and collaboration, this system ensures seamless file handling while prioritizing accessibility, security, and ease of use.

The primary objectives of this system include:

- **Secure File Storage** – Users can upload and store various file types, including documents, images, videos, and more, in a structured and accessible manner.
- **Efficient File Management** – Features such as file preview, renaming, deletion, sorting, and searching allow users to organize and retrieve files easily.
- **Seamless File Sharing** – Users can share files directly by entering the recipient's email, eliminating the need for external sharing links.
- **User Authentication and Access Control** – The system ensures that only authorized users can access, manage, and share their files, maintaining data security and privacy.
- **Real-Time Updates and Notifications** – Instant feedback mechanisms and email notifications keep users informed about file-sharing activities and system updates.
- **Responsive and Intuitive User Interface** – A modern, responsive design ensures accessibility across devices, improving the user experience.

This system is built using **React 19, Next.js 15, Appwrite, TypeScript, TailwindCSS, and ShadCN**, ensuring high performance, scalability, and a smooth user experience. By streamlining storage and sharing functionalities, it caters to individuals and teams looking for a reliable file management solution.

Scope:

The scope of the Storage and File Sharing Platform extends across various user needs, from

individuals managing personal files to businesses handling large-scale data sharing. It is designed to offer a secure, scalable, and intuitive file management solution, capable of handling high volumes of uploads, downloads, and shared content. With integrated global search, sorting, and a dynamic dashboard, users can efficiently organize and retrieve their files. The platform's responsive design ensures accessibility across desktop, tablet, and mobile devices for seamless user experience.

1.4 Achievements

- Built a scalable and efficient platform capable of handling multiple users and large file uploads simultaneously.
- Implemented a secure file-sharing system, allowing users to share files via unique links while maintaining access control.
- Developed an intuitive and responsive UI, ensuring seamless accessibility across desktops, tablets, and mobile devices.
- Integrated real-time global search and sorting features, enabling users to find and organize files quickly.
- Enhanced system reliability by implementing error tracking and monitoring tools to identify and resolve issues proactively.
- Optimized storage management, allowing users to view their total storage usage, recent uploads, and file breakdowns efficiently.

CHAPTER NO. 02

SURVEY OF TECHNOLOGY

2.1 Technologies Used

- Next.js 15: A modern framework for building high-performance, server-side rendered applications with optimized loading speeds and improved SEO.
- Appwrite: A backend-as-a-service (BaaS) solution providing user authentication, database management, and cloud storage for handling file uploads securely.
- TypeScript: A statically typed superset of JavaScript, ensuring type safety, scalability, and maintainability for the codebase.
- TailwindCSS: A utility-first CSS framework that enables fast and responsive UI design with minimal effort.
- ShadCN: A component library offering pre-built, customizable UI components, making development faster and more efficient.
- Twilio: An API service integrated for SMS notifications, ensuring timely communication regarding file-sharing activities or account updates.
- Cloud Storage: Secure, scalable storage for managing uploaded files, user data, and shared content efficiently.

2.2 React 19 & Next.js 15

Next.js

History: Next.js was created by Vercel (formerly Zeit) and launched in 2016. It was built as a React framework that provides features like server-side rendering (SSR) and static site generation (SSG). These features address performance bottlenecks in traditional React applications, improving load times and SEO. Next.js is highly optimized for modern web development, especially for applications that need to perform well across different devices and networks.

Implementation in the Project: Next.js 15 serves as the backbone of the Storage and File Sharing Platform, managing the front-end UI and API interactions. It enables fast rendering, improved caching, and optimized loading of file-sharing components, ensuring a smooth user experience.

Key Features Used:

- Server-Side Rendering (SSR): Enhances performance by preloading file data.
- Static Site Generation (SSG): Optimizes the UI for non-dynamic pages.
- API Routes: Handles backend operations like file uploads, user authentication, and data retrieval.
- Image Optimization: Ensures fast loading of thumbnails and previews.

React 19

History: React, developed by Meta (formerly Facebook), was first released in 2013 as a JavaScript library for building user interfaces. It revolutionized front-end development with its component-based architecture and virtual DOM for efficient rendering. Over the years, React has evolved significantly, introducing features like Hooks, Concurrent Mode, and Server Components to enhance performance and developer experience. React 19, the latest version, continues this trend by improving rendering efficiency, optimizing hydration, and introducing new primitives for building interactive applications.

Implementation in the Project: React 19 plays a crucial role in the Storage and File Sharing Platform, providing a fast, interactive, and modular front-end experience. With its optimized rendering capabilities, React 19 ensures smooth user interactions while handling file uploads, previews, and sharing actions efficiently. The latest advancements help in minimizing re-renders, improving performance, and enhancing the scalability of the platform.

Key Features Used:

- React Server Components (RSC): Improves performance by offloading rendering to the server.
- Concurrent Rendering: Enhances UI responsiveness, ensuring smooth file navigation.

- useOptimistic Hook: Provides immediate feedback on actions like file uploads and renaming.
- Automatic Batching: Reduces unnecessary re-renders, leading to a more efficient UI.
- Improved Hydration: Ensures faster loading and seamless interaction after initial page load.

By leveraging React 19, the platform delivers a modern, highly optimized user experience, making file management effortless and efficient.

2.3 Appwrite Backend Service

History: Appwrite, first introduced in 2020, is an open-source backend-as-a-service (BaaS) platform that simplifies the development of modern web and mobile applications. It provides developers with ready-to-use backend services such as authentication, database, storage, and serverless functions. Appwrite focuses on providing an all-in-one backend solution that can be hosted by developers themselves, giving them more control over their data and infrastructure.

Implementation in PMS: Appwrite is a core component of the **Storage and File Sharing Platform**, responsible for managing user authentication, file storage, and database operations. It provides a secure and scalable backend, allowing users to register, log in, upload files, and share them with appropriate access permissions. By leveraging Appwrite's built-in security features, the platform ensures encrypted data handling, seamless authentication, and structured metadata storage.

Key Features Utilized

1. User Authentication

- Implemented Appwrite Auth to enable secure user registration and login.
- Supports multiple authentication methods, such as email/password, OAuth (Google, GitHub, etc.), and anonymous login.
- Utilizes JWT-based session management to ensure secure access to user-specific files.
- Provides session persistence and automatic token refresh for a smooth user experience.

2. Database Management

- Uses Appwrite's Database service to store and manage:
    - File metadata (e.g., file name, size, type, and upload timestamp).
    - User profiles, including preferences and account settings.
    - Access control lists (ACLs) to define file-sharing permissions.
- Ensures data integrity and structured querying for efficient file retrieval and access control.

3. Cloud Storage & File Management

- Utilizes Appwrite Storage for handling file uploads and downloads securely.
- Supports role-based access permissions, enabling users to:
    - Keep files private (accessible only to the owner).
    - Share files with specific users or user groups.
    - Make files publicly accessible via a secure link.
- Implements file versioning to allow users to track changes and restore previous versions if needed.
- Enables real-time updates and event-based triggers (e.g., notifications when a file is shared or modified).

By leveraging these features, the platform provides a seamless and secure file storage experience, ensuring reliable authentication, structured data management, and flexible access control mechanisms.

2.4 Typescript

History: TypeScript was developed by Microsoft and released in 2012 as a statically typed superset of JavaScript. Its main purpose is to enhance JavaScript’s capabilities by introducing static types, allowing for earlier detection of errors in code. TypeScript's strong typing system makes large applications more scalable and maintainable, which is particularly useful in enterprise-level software.

TypeScript is a fundamental part of the **Storage and File Sharing Platform**, ensuring **code reliability, maintainability, and scalability** throughout development. By enforcing static typing, TypeScript helps prevent common runtime errors, making the platform more robust and easier to manage as it grows. It is used extensively across **API integrations, user authentication, file management, and database operations**, providing clear type definitions and structured data handling.

Key Benefits in the Project

1. Type Safety

- Enforces strict typing for key entities such as users, files, API responses, and access controls.
- Reduces the chances of undefined or incorrect values being passed to functions or components.
- Helps catch type-related errors during compilation rather than at runtime, improving overall application stability.

2. Improved Debugging & Maintainability

- Identifies potential bugs early in the development cycle, minimizing production issues.
- Provides detailed error messages and clear stack traces, making debugging more efficient.
- Ensures consistency across components and API interactions, reducing unexpected behavior.

3. Enhanced IDE Support & Developer Experience

- Enables better auto-completion, inline documentation, and intelligent suggestions in VS Code and other IDEs.
- Simplifies code refactoring by allowing safe changes to interfaces, types, and functions without breaking the application.
- Improves collaboration among developers by providing self-documenting code with explicit type definitions.

2.5 TailwindCSS & ShadCN

TailwindCSS

TailwindCSS is a **utility-first CSS framework** that enables developers to build modern UIs quickly without writing extensive custom CSS. It provides **low-level utility classes** for styling elements directly in the markup, allowing for fast prototyping, a consistent design system, and a highly maintainable codebase.

Implementation in the Project: In the Storage and File Sharing Platform, TailwindCSS is used to design a modern, responsive, and minimalistic user interface. Its utility classes allow for a highly flexible and scalable design system while keeping the styles consistent across different components. By eliminating the need for separate stylesheets, TailwindCSS speeds up development and reduces CSS bloat.

Key Features Used

1. Responsive Design
    - Uses mobile-first breakpoints (sm, md, lg, xl, 2xl) to ensure the UI adapts seamlessly across different devices.
    - Provides grid and flexbox utilities for dynamic and responsive layouts.
    - Enables dark mode support for a better user experience.
2. Utility-First Approach
    - Eliminates the need for writing custom CSS by using predefined utility classes.
    - Allows for inline styling directly in the JSX/TSX components.
    - Encourages consistency in UI design without global styles interfering.
3. Theme Customization
    - Uses Tailwind's configuration file (tailwind.config.js) to customize themes, including colors, typography, spacing, and animations.
    - Supports custom design tokens, making branding and UI adjustments easy.

- Enables variants like hover, focus, and active states for interactive elements.

By leveraging TailwindCSS, the platform ensures a scalable, maintainable, and visually appealing design while keeping performance optimized.

ShadCN

History: ShadCN is a modern component library that leverages the power of TailwindCSS. It was designed to work alongside Tailwind to provide out-of-the-box components that can be easily customized. ShadCN simplifies the process of building complex user interfaces by offering pre-built components that can be quickly styled to fit the project's needs.

Implementation in the Project: ShadCN is used to create reusable UI elements such as file upload buttons, modals, dashboard widgets, and authentication forms. Its integration with TailwindCSS ensures a seamless design system.

Key Features Used:

- **Pre-built Components:** Reduces development time with ready-to-use UI elements.
- **Customization:** Easily styled using TailwindCSS.
- **Accessibility:** Ensures compliance with web accessibility standards.

CHAPTER NO. 03

REQUIREMENT AND ANALYSIS

3.1 Problem Statement

Managing and sharing digital files can be inefficient and disorganized, especially when dealing with large volumes of data across multiple devices. Many existing solutions lack a user-friendly interface, making it difficult to upload, locate, and share files effortlessly. Additionally, security concerns arise with unauthorized access to sensitive files. This platform addresses these challenges by providing a structured, secure, and efficient file management system, allowing users to store, organize, and share files with ease while maintaining full control over their data.

3.2 Requirement Specification

This section outlines the functional and non-functional requirements necessary for the system's development and operation.

Functional Requirements

These define the core functionalities of the system:

- User Authentication: Users should be able to sign up, log in, and log out securely using Appwrite authentication.
- File Upload and Management: Users must be able to upload various file types (documents, images, videos, etc.), rename files, and delete them when needed.
- File Download: Users should be able to download their uploaded files at any time.
- File Sharing: Users can share files by entering the recipient's email, enabling direct file transfer.
- Search and Sort: Users can search for files using keywords and sort them based on name, date, or size.
- Dashboard Overview: The system provides a summary of storage usage, recent uploads, and file types.

- Access Control: Only authorized users should be able to access and manage their own files.

Non-Functional Requirements

These define system performance and operational constraints:

- Security: All file uploads, downloads, and user data should be securely stored and transmitted using encryption.
- Scalability: The system should efficiently handle multiple users and large file uploads without performance degradation.
- Performance: The platform should load quickly and provide a smooth user experience with minimal latency.
- Reliability: The system must ensure high availability and prevent data loss through regular backups.
- User-Friendly Interface: A clean, responsive, and intuitive UI should make file management easy across all devices.

By meeting these requirements, the platform ensures a secure, scalable, and user-friendly file-sharing experience for all users.

3.3 Planning & Scheduling

Planning is the process of setting goals, defining strategies, and outlining tasks and schedules to achieve specific objectives. It ensures a structured approach to project execution by identifying key milestones and deliverables.

Scheduling involves allocating specific time slots for tasks, activities, or events to ensure they are completed in an organized and timely manner. It helps manage resources efficiently and keeps the project on track.

A **PERT** chart maps task sequences and dependencies to analyze project timelines and identify the critical path.

A **Gantt** chart visually represents project tasks on a timeline, helping track progress and manage deadlines.

3.3.1 Gantt Chart

3.3.2 PERT Chart

3.4 Software and Hardware Requirements

The **Software and Hardware Requirements** are essential for the development, execution, and maintenance of the Storage and File Sharing Platform. Hardware provides the necessary computational power, while software ensures smooth implementation and operation of the system. Without the right combination of both, the system cannot function optimally.

Hardware Requirements

Hardware defines the physical components needed to support the system. The performance and efficiency of the platform depend on these specifications.

- **Device Type**: Laptop
- **RAM**: 16 GB
- **Storage**: 512 GB SSD
- **Processor**: 12th Gen Intel® Core™ i5-1240P (1.70GHz)

Software Requirements

Software includes essential tools, frameworks, and dependencies required for development and execution. It ensures the system functions as intended and supports user interactions.

1. **Operating System Compatibility**
    - Linux
    - macOS
    - Windows 11
2. **Database Used**
    - Appwrite (for storage and authentication)
3. **Development Tools and Frameworks**
    - **Next.js 15** (Front-end and API handling)
    - **React 19** (User interface development)

- **TypeScript** (Type-safe coding)
- **TailwindCSS** (Styling framework)
- **ShadCN** (Component library)
- **Node.js & npm** (Runtime environment and package management)

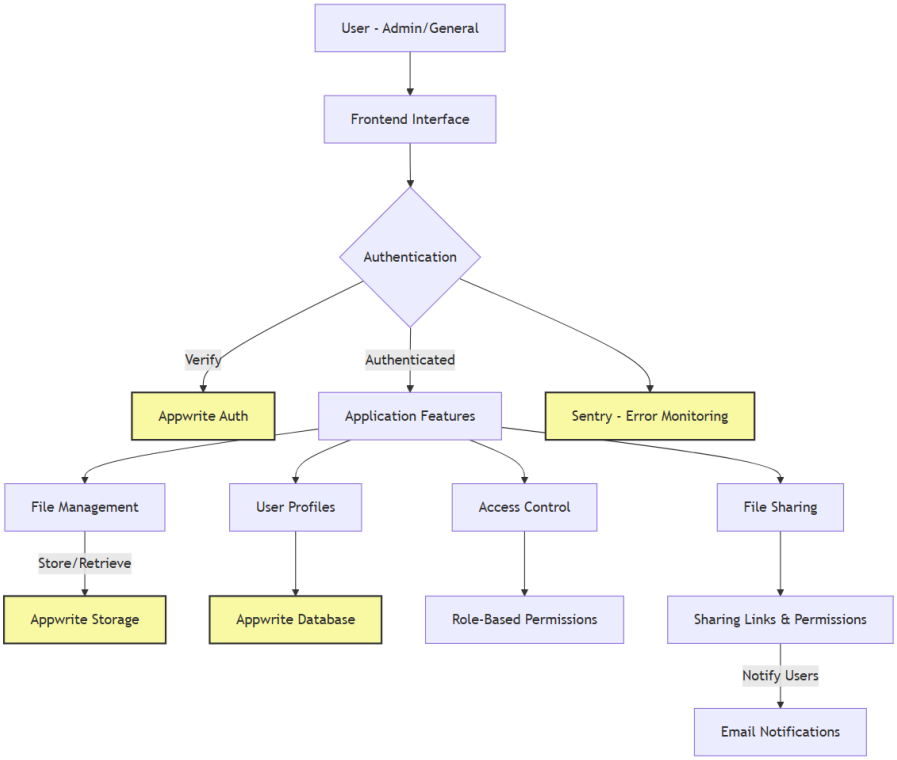
These hardware and software components collectively provide a strong foundation for building and running the Storage and File Sharing Platform efficiently.

3.5 Preliminary Product Description

1. **Users** – Users are the core entities of the system, enabling authentication, file management, and sharing functionalities. They can upload, organize, and share files effortlessly.
2. **File Upload** – Users can upload various file types, including documents, images, videos, and audio, ensuring secure storage and easy access.
3. **File Management** – Users can rename, delete, and organize files into different categories for efficient storage and retrieval.
4. **File Sharing** – The platform allows users to share files directly by entering the recipient's email, making collaboration seamless.
5. **Dashboard** – Provides an overview of storage usage, recent uploads, and file categorization, helping users track their data efficiently.
6. **Search & Sorting** – Users can quickly locate files using a global search feature and organize them based on name, size, or date.
7. **Security & Authentication** – Appwrite authentication ensures secure access with login and logout features, protecting user data.
8. **Cross-Device Compatibility** – The platform is fully responsive, allowing users to access and manage their files across desktops, tablets, and mobile devices.

3.6 Conceptual Models

3.6.1 Application Structure



The **Storage and File Sharing Platform** follows a modular and component-based architecture, ensuring scalability, flexibility, and maintainability. Each module is independently structured, focusing on a separation of concerns for smooth system operation.

User Authentication

This module handles user registration, authentication, and profile management. Built using Next.js and Appwrite, it ensures secure login functionality while storing user credentials and session data in the backend. Authentication methods include email/password, OAuth, and token-based authentication.

File Upload

Users can upload files securely using this module, which integrates Appwrite storage for efficient handling of file uploads. The system validates file types, sizes, and integrity before processing uploads. Progress indicators and error handling mechanisms ensure a smooth upload experience.

File Storage & Management

This module provides a secure storage system where users can view, organize, and manage their files. It supports file categorization, metadata tagging, and version control, ensuring users can track changes and manage their digital assets effectively.

File Sharing & Permissions

Users can share files with specific individuals or groups by setting appropriate permissions and access levels. This module enforces role-based access control (RBAC) to ensure only authorized users can view, edit, or download shared files.

User Interface

Built using React 19, TailwindCSS, and ShadCN, this module provides a clean, responsive, and user-friendly interface. Users can drag-and-drop files, navigate intuitive dashboards, and access their storage efficiently across different devices.

Security & Access Control

To protect user data, this module implements encryption, authentication layers, and session management. It ensures that stored files remain private, secure, and accessible only to authorized users.

Real-Time Notifications

Powered by Appwrite's event-driven architecture, users receive real-time alerts for file uploads, shares, edits, and deletions. This ensures seamless collaboration and instant updates on shared files.

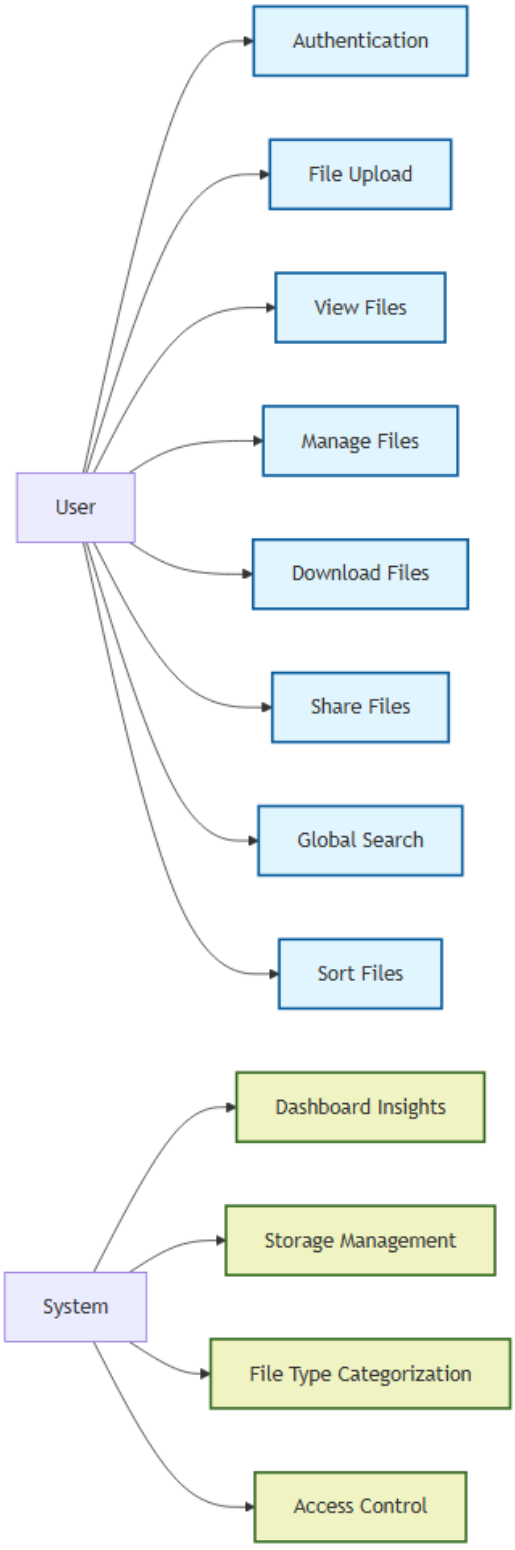
Admin & Monitoring Panel

This module provides administrators with a dashboard to monitor user activities, storage usage, and access logs. Admins can set policies, manage user roles, and track file-sharing trends for better oversight.

3.6.2 Event Table

Event Name	Trigger	Source	Response	Destination	Data Stored/Updated	Notification Sent
User Registration	User submits form	Registration Page	Creates user account and stores data	User Dashboard	User credentials, profile info	Welcome email
File Upload	User selects and uploads a file	Upload Button	Stores file in Appwrite and updates UI	File List Page	File metadata (name, size, type)	None
File Download	User clicks download	File List Page	Fetches file and starts download	User's Device	Download count updated	None
File Sharing	User enters recipient email and sends file	File Options Menu	Sends file directly to recipient	Recipient's Email	Shared file record updated	Recipient's Profile UI
File Deletion	User deletes a file	File Options Menu	Removes file from storage and updates UI	File List Page	File removed from database	None
User Logout	User clicks logout	Dashboard Page	Ends session and redirects to login	Login Page	Session data cleared	None

3.6.3 Use Case Diagram



The **Use Case Diagram** represents the key interactions between users and the Storage and File Sharing Platform, ensuring seamless file management, controlled sharing, and administrative oversight.

Actors & Their Roles:

User (Registered/Guest)

- Can register/login use authentication.
- Upload, store, and manage files securely.
- Share files via email sent to the user and opens up to their UI.
- Modify access controls, allowing view-only or edit permissions.
- Delete files permanently when needed.

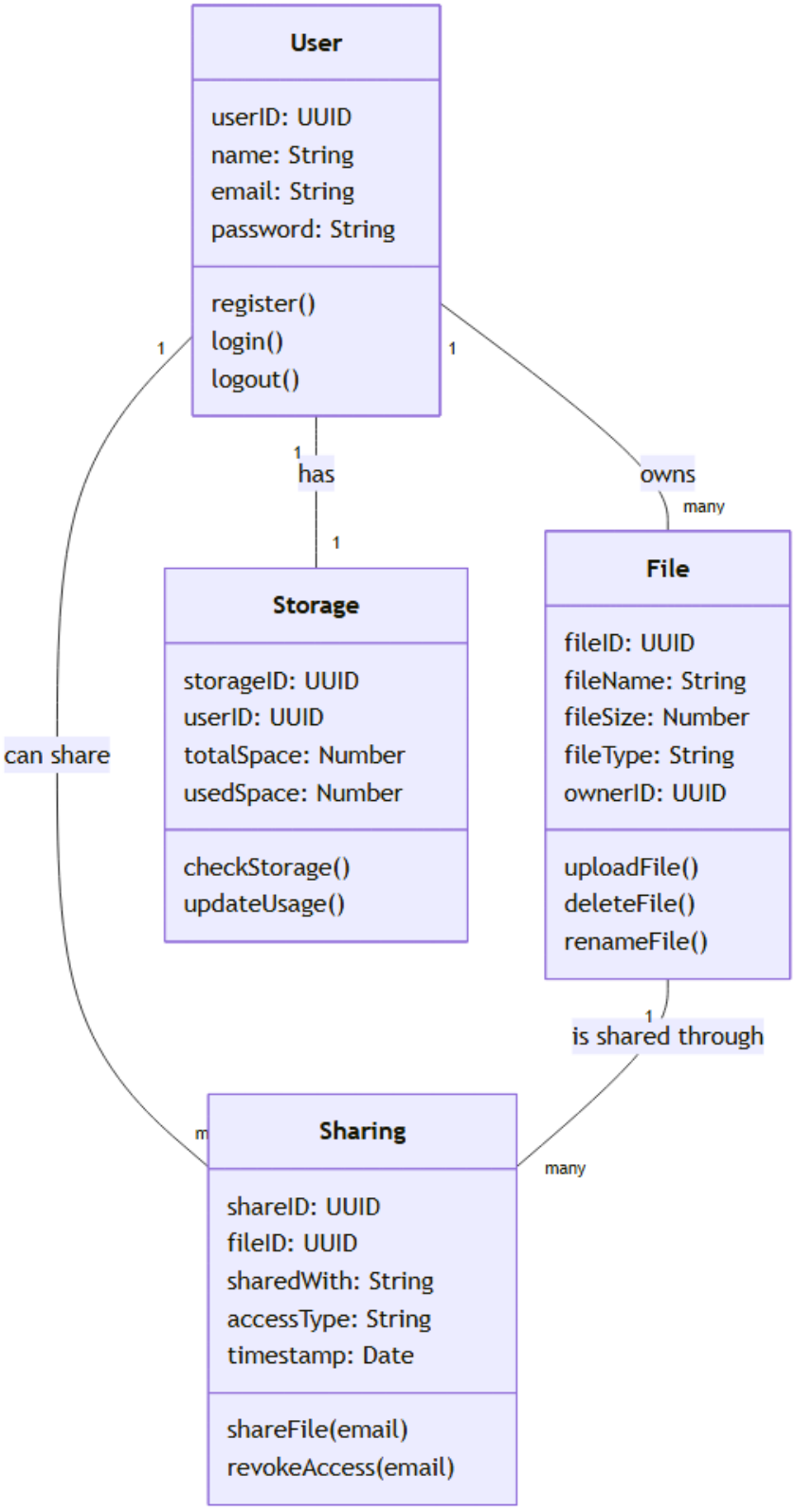
Admin User

- Manages user accounts, ensuring authorized access.
- Monitors storage usage and enforces usage policies.
- Oversees shared files to detect policy violations.
- Enforces security controls, preventing unauthorized access.
- Removes flagged or inappropriate content based on reports.

Key System Features:

- Secure Authentication & Access Control via Appwrite Auth.
- Efficient File Storage & Retrieval using Appwrite Storage.
- Real-time Database Updates for user and admin actions via Appwrite Database.
- Seamless File Sharing with controlled permissions.
- Administrative Oversight for security & content management.

3.6.4 Class Diagram



The class diagram represents the main entities in the **Storage and File Sharing System** and their relationships:

User Class

- Attributes: userID, name, email, password
- Methods: register(), login(), logout()
- Description: Represents a user in the system who can upload, delete, and share files. Each user has a unique ID, and their authentication details are stored securely.

File Class

- Attributes: fileID, fileName, fileSize, fileType, ownerID
- Methods: uploadFile(), deleteFile(), renameFile()
- Description: Represents the files stored in the system. Each file is associated with a user (owner), and users can perform operations such as uploading, deleting, and renaming files.

Sharing Class

- Attributes: shareID, fileID, sharedWith, accessType, timestamp
- Methods: shareFile(email), revokeAccess(email)
- Description: Manages file-sharing functionality. A file can be shared with specific users via email, granting them defined access permissions. Users can also revoke access when needed.

Storage Class

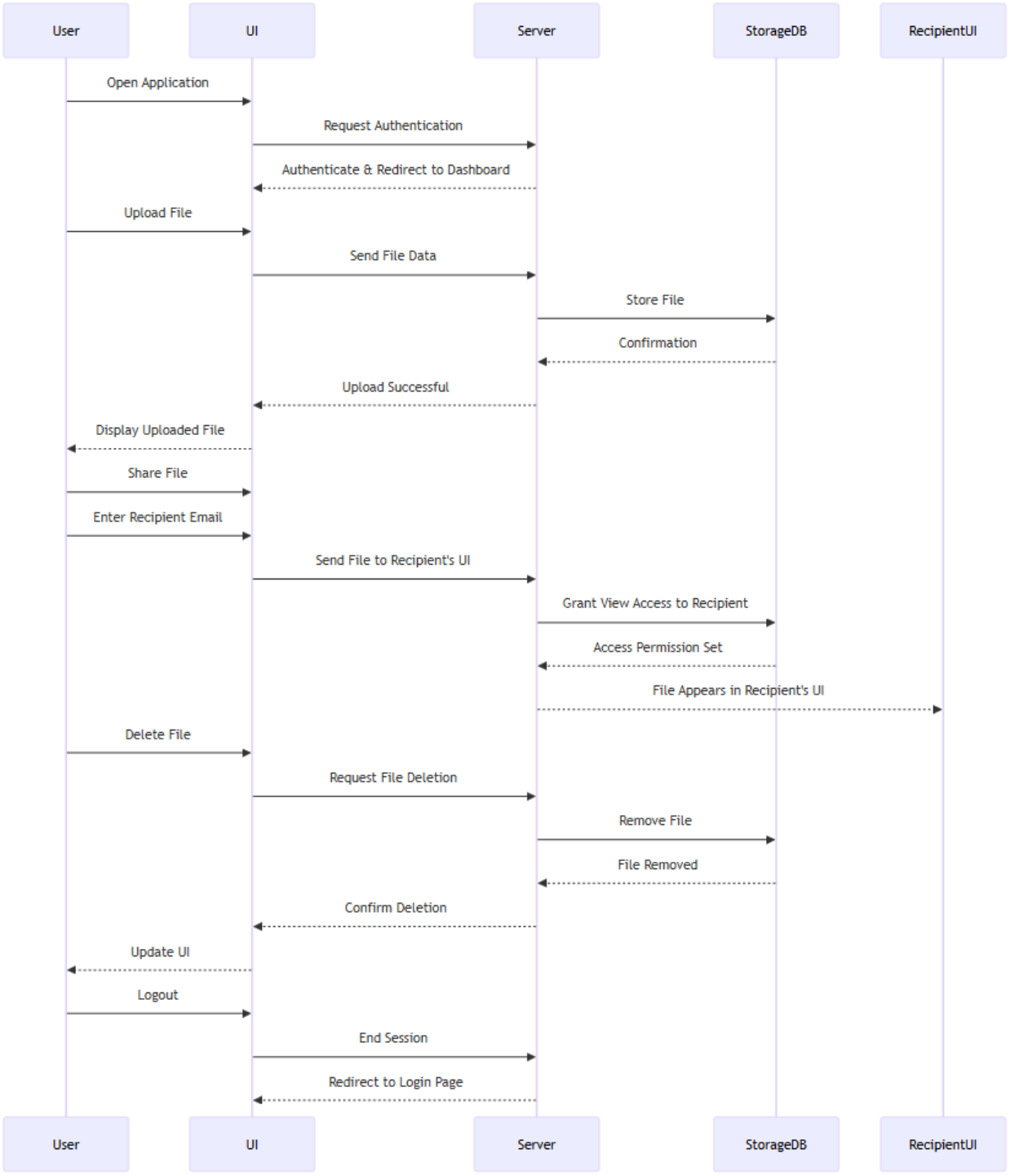
- Attributes: storageID, userID, totalSpace, usedSpace
- Methods: checkStorage(), updateUsage()
- Description: Manages the available and used storage for each user. It keeps track of storage limits and updates the usage details when files are uploaded or deleted.

Class Relationships

- User "1" -- "many" File: A user can own multiple files.
- User "1" -- "many" Sharing: A user can share multiple files with different users.
- File "1" -- "many" Sharing: A file can be shared with multiple recipients.
- User "1" -- "1" Storage: Each user has a dedicated storage allocation.

This class diagram ensures that the system's entities and their behaviors are well-structured, enabling efficient file management, access control, and user interaction.

3.6.5 Sequence Diagram



The **sequence diagram** illustrates the interaction between the User, Frontend UI, and various backend services such as Authentication, Database, Storage, and Notification Service in your Storage and File Management Platform. Below is a detailed breakdown of each step in the process.

1. User Authentication Flow

Purpose: Ensures secure access to the platform.

1. User → Frontend UI: The user accesses the system.
2. Frontend UI → Auth Service: The frontend sends login credentials for verification.
3. Auth Service → Frontend UI: If valid, an authentication token is returned, granting access to the user.

2. File Upload Process

Purpose: Allows users to upload files securely.

1. User → Frontend UI: The user selects a file and initiates an upload.
2. Frontend UI → Storage Service: The file is validated and uploaded to Appwrite Storage.
3. Storage Service → Database: Metadata of the uploaded file (such as file name, size, type, and ownership) is stored in Appwrite Database.
4. Database → Frontend UI: Confirms the successful storage of metadata.
5. Frontend UI → User: Notifies the user that the file has been uploaded successfully.

3. File Sharing Process

Purpose: Enables users to share files securely with specific recipients.

1. User → Frontend UI: The user selects a file and enters the recipient's email.
2. Frontend UI → Server: The system sends a request to share the file with the provided email.
3. Server → Database: The system updates permissions to allow access for the recipient.
4. Database → Server: Confirms the sharing action.
5. Server → Frontend UI: Notifies the user that the file has been successfully shared.

6. Frontend UI → Recipient: The recipient receives an email with access to the shared file.

4. Access Control Management

Purpose: Allows users to manage permissions for their files.

1. User → Frontend UI: The user modifies file access permissions (public/private, role-based access).
2. Frontend UI → Database: Updates the file permissions accordingly.
3. Database → Frontend UI: Confirms the successful update.

5. File Deletion Process

Purpose: Enables users to delete unwanted files from the system.

1. User → Frontend UI: The user selects a file and requests deletion.
2. Frontend UI → Storage Service: The system removes the file from storage.
3. Storage Service → Database: The corresponding metadata is also deleted.
4. Database → Frontend UI: Confirms that the file and its metadata have been removed.
5. Frontend UI → User: Notifies the user that the file has been deleted successfully.

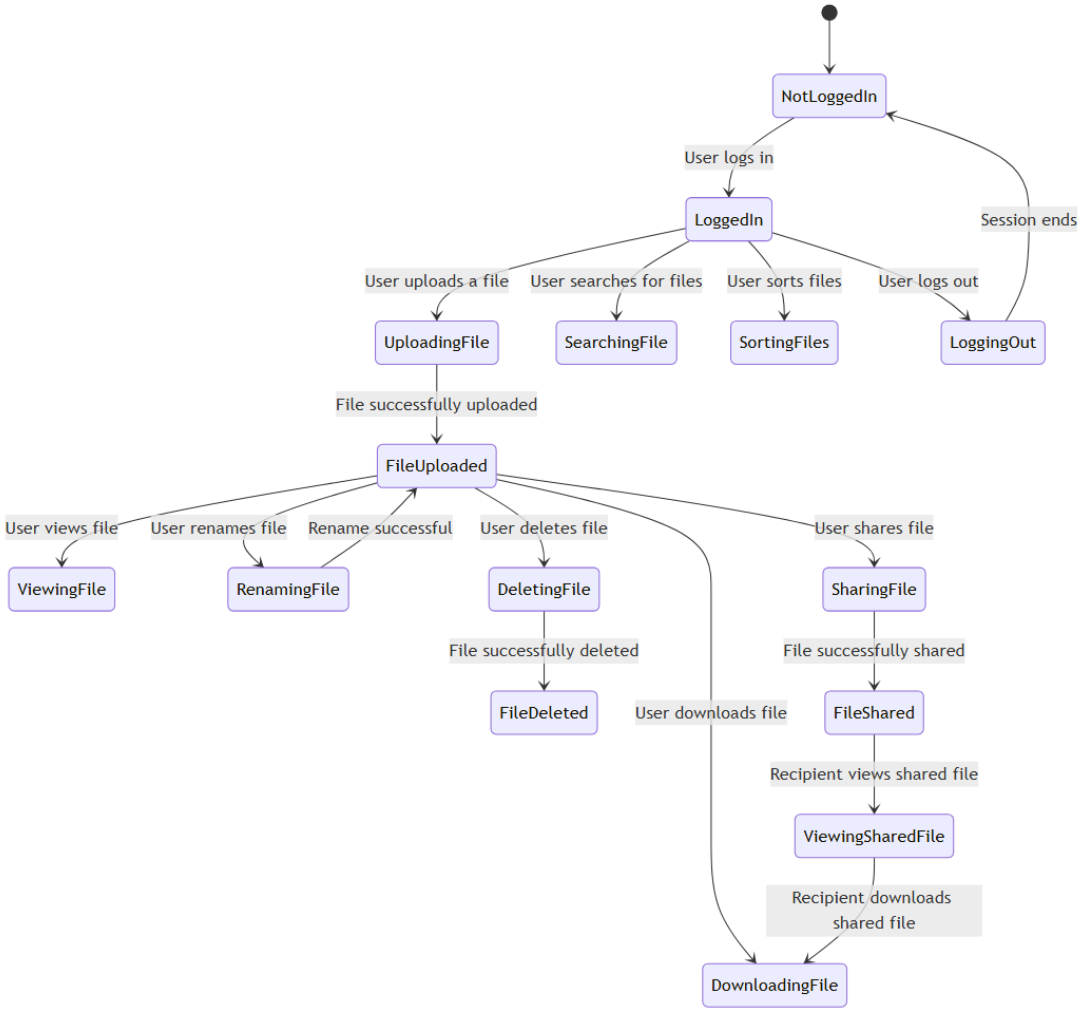
6. Notification System

Purpose: Sends real-time updates and confirmations to users.

1. Database → Notification Service: Triggers a notification whenever a file is uploaded, shared, or deleted.

Notification Service → User: Sends a confirmation message (e.g., "Your file has been successfully uploaded.")

3.6.6 State Machine Diagram

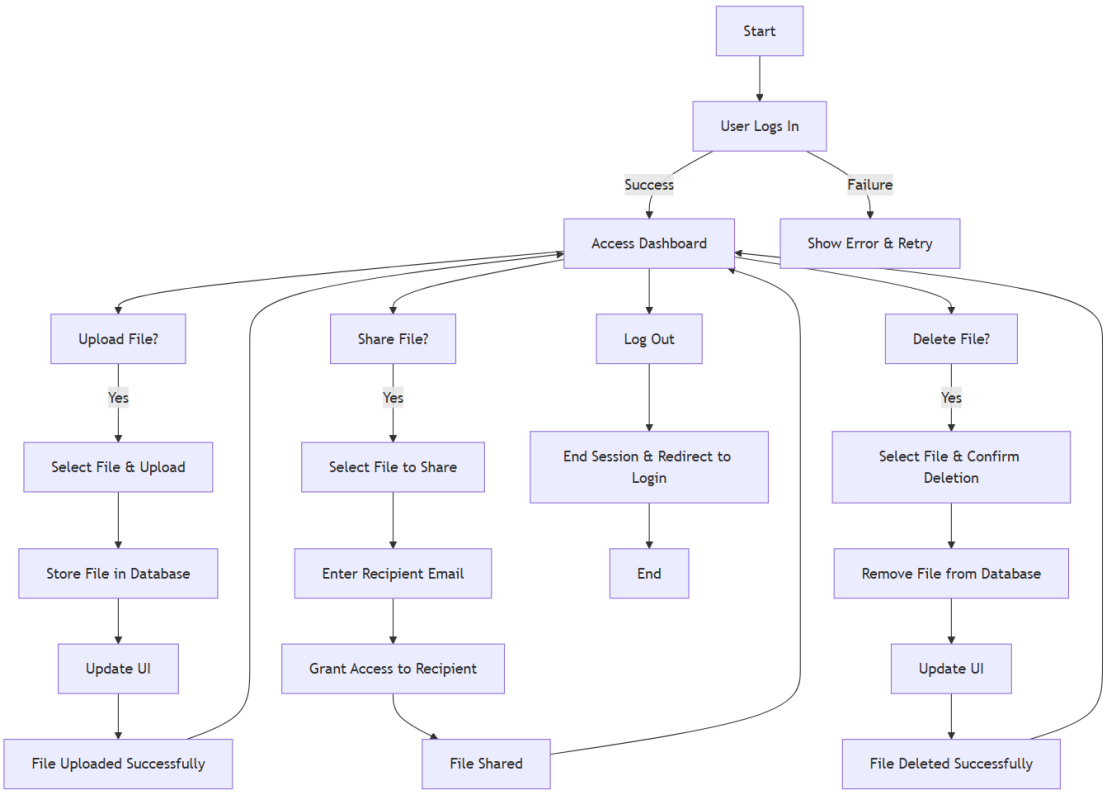


The state machine diagram represents the different states that a file can go through in the storage and file management platform:

- **Unauthenticated:**
  When a user opens the app without logging in, they remain in the unauthenticated state until they sign in.
- **Authenticated:**
  After successful authentication, the user gains access to file management features like uploading, sharing, and deleting files.
- **Uploading:**
  When a user selects a file to upload, the state changes to "uploading." If successful, it transitions to "upload success"; otherwise, it moves to "upload failed," allowing a retry.
- **Sharing:**
  If the user shares a file, the state moves to "sharing." A successful share is done and shown to the receiver's UI and can perform activities.
- **Deleting:**
  When a user deletes a file, the state moves to "deleting." If successful, the file is removed; otherwise, the user can attempt deletion again.
- **Logging Out:** When the user logs out, the system transitions back to the unauthenticated state, ending the session.

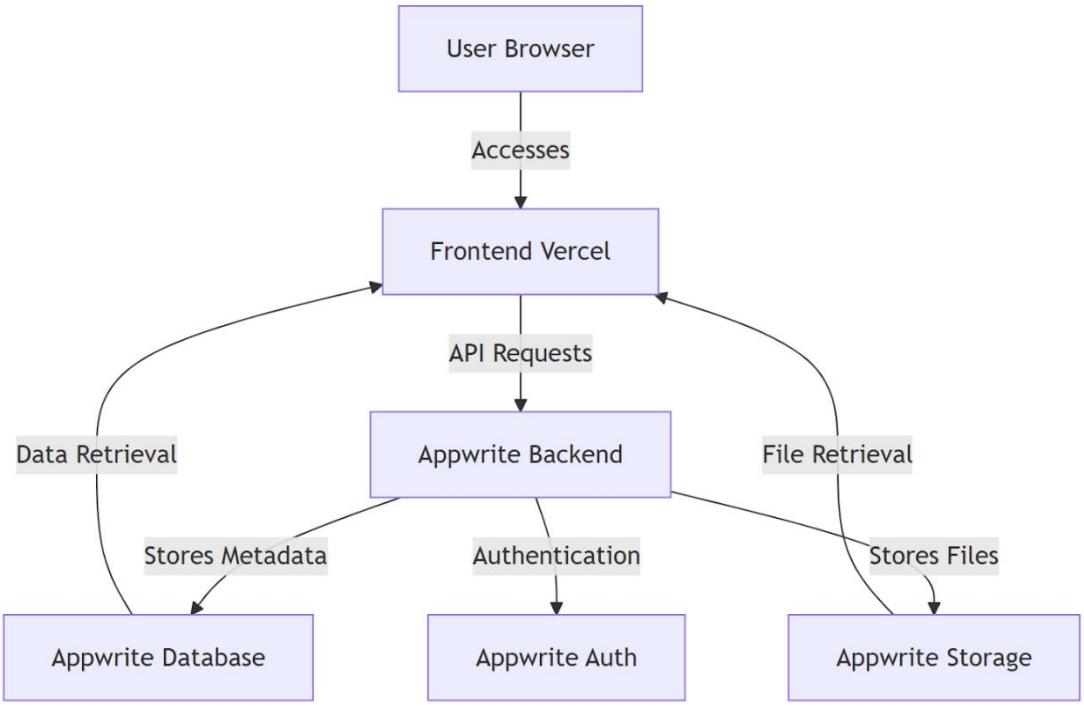
This state machine ensures seamless file management, tracking each action from authentication to file operations.

3.6.7 Activity Diagram



1. **User Uploads a File:** The user logs into the system and selects a file to upload.
2. **System Processes the Upload:** The system verifies the file type, size, and user permissions before initiating the upload.
3. **File Storage & Database Update:** Once verified, the file is stored in Appwrite Storage, and the system updates the Appwrite Database with metadata, including the filename, size, and access permissions.
4. **User Manages Files:** Users can view, rename, delete, or set access controls on uploaded files.
5. **File Sharing Process:** Users can generate sharing links and define access permissions (public, private, or restricted).
6. **Notification & External Services:** If a file is shared, the system triggers a notification (email/SMS) to inform the recipient about the shared file.

3.6.8 Deployment Diagram



The deployment diagram illustrates how the storage and file-sharing platform is hosted and accessed:

- The frontend (React and Next.js) is deployed on Vercel, ensuring fast content delivery and scalability.
- The backend (Appwrite) handles authentication, database storage, and file management, interacting with the frontend through API requests.
- Appwrite Database stores metadata, while Appwrite Storage manages file uploads and retrievals.

This architecture ensures efficient data handling, seamless authentication, and secure file storage with minimal latency for users.

3.6.9 SWOT Analysis

SWOT analysis helps in assessing the strengths, weaknesses, opportunities, and threats associated with the **Storage and File Sharing Platform** project. This analysis provides insights for strategic improvements and risk mitigation.

CHAPTER NO. 04

SYSTEM DESIGN

4.1 Basic Modules

User Management Module

- Handles user registration, login, and authentication.
- Manages user profiles and account settings.

File Management Module

- Allows users to upload, rename, and delete files.
- Maintains metadata such as file type, size, and owner.

File Sharing Module

- Enables users to share files by entering the recipient's email.
- Grants appropriate access permissions for shared files.

Storage Management Module

- Tracks storage usage for each user.
- Ensures storage limits and optimizes file storage.

Security & Access Control Module

- Implements authentication and authorization.
- Ensures secure file access with proper permissions.

User Interface (UI) Module

- Provides an interactive dashboard for users.
- Displays files, shared items, and storage usage.

4.2 Data Design

The database design transforms the logical data model into a structured database definition, ensuring efficient storage, retrieval, and security. It incorporates constraints and relationships to maintain data integrity.

4.2.1 Schema Design with Data Integrity and Constraints

- Primary Key Constraint: Ensures each record in a table is uniquely identified, preventing duplicate entries and enforcing data consistency.
- Foreign Key Constraint: Establishes relationships between tables, ensuring referential integrity by preventing orphan records.
- Unique Key Constraint: Ensures specific columns contain unique values, preventing duplicate entries in a table.
- Not Null Constraint: Ensures that a column cannot have NULL values, enforcing mandatory data entry.
- Auto Increment: Automatically generates a unique identifier for new records, ensuring sequential and unique primary key values.

4.2.2 Database Relationship Design

The **Storage and File Sharing Platform** maintains structured relationships between various entities to ensure efficient file storage, sharing, and user authentication. The database is designed with constraints and relationships to maintain integrity.

1. Users Table

Stores user information, including authentication credentials.

```
CREATE TABLE users (
    user_id UUID PRIMARY KEY,
    full_name VARCHAR(255) NOT NULL,
```

```
    email VARCHAR(255) UNIQUE NOT NULL,
    account_id UUID UNIQUE NOT NULL,
    avatar_url VARCHAR(255) DEFAULT 'https://example.com/avatar.png',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- **Primary Key:** user_id
- **Unique Constraints:** email, account_id
- **Relationships:** Connected to the **Files** table via owner_id

2. Files Table

Stores metadata of uploaded files and links each file to an owner.

```
CREATE TABLE files (
    file_id UUID PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    extension VARCHAR(50) NOT NULL,
    file_size INT NOT NULL,
    file_type VARCHAR(50) NOT NULL,
    owner_id UUID NOT NULL,
    bucket_file_id UUID UNIQUE NOT NULL,
    uploaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (owner_id) REFERENCES users(user_id) ON DELETE CASCADE
);
```

- **Primary Key:** file_id

- **Foreign Key:** owner_id → users(user_id)
- **Unique Constraints:** bucket_file_id
- **Relationships:**
    - Connected to the **Users** table (owner of the file)
    - Connected to the **Sharing** table (for shared access)

3. Storage Table

Tracks storage limits and usage for each user.

```
CREATE TABLE storage (
   storage_id UUID PRIMARY KEY,
   user_id UUID UNIQUE NOT NULL,
   total_space INT NOT NULL DEFAULT 2147483648, -- 2GB in bytes
   used_space INT NOT NULL DEFAULT 0,
   FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
);
```

- **Primary Key:** storage_id
- **Foreign Key:** user_id → users(user_id)
- **Unique Constraint:** user_id
- **Relationships:**
    - Connected to the **Users** table (tracks each user's storage usage)

4. Sharing Table

Manages file-sharing permissions between users.

```
CREATE TABLE sharing (
    share_id UUID PRIMARY KEY,
    file_id UUID NOT NULL,
    shared_with UUID NOT NULL,
    access_type VARCHAR(50) CHECK (access_type IN ('view', 'edit')),
    shared_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (file_id) REFERENCES files(file_id) ON DELETE CASCADE,
    FOREIGN KEY (shared_with) REFERENCES users(user_id) ON DELETE CASCADE
);
```

- **Primary Key:** share_id
- **Foreign Keys:**
    - file_id → files(file_id)
    - shared_with → users(user_id)
- **Constraints:** access_type must be either "view" or "edit"
- **Relationships:**
    - Connected to **Users** (who the file is shared with)
    - Connected to **Files** (which file is shared)

5. Permissions Table

Defines user-specific access control for files.

```
CREATE TABLE permissions (
    permission_id UUID PRIMARY KEY,
    file_id UUID NOT NULL,
```

```
    user_id UUID NOT NULL,
    access_type VARCHAR(50) CHECK (access_type IN ('view', 'edit', 'delete')),
    FOREIGN KEY (file_id) REFERENCES files(file_id) ON DELETE CASCADE,
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
);
```

- **Primary Key:** permission_id
- **Foreign Keys:**
  - file_id → files(file_id)
  - user_id → users(user_id)
- **Constraints:** access_type must be either "view", "edit", or "delete"
- **Relationships:**
  - Connected to **Files** (which file the user has access to)
  - Connected to **Users** (who has permission for the file)

6. Audit Logs Table

Tracks file actions such as uploads, renames, deletions, and sharing.

```
CREATE TABLE audit_logs (
    log_id UUID PRIMARY KEY,
    file_id UUID NOT NULL,
    user_id UUID NOT NULL,
    action VARCHAR(50) CHECK (action IN ('upload', 'rename', 'delete', 'share')),
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

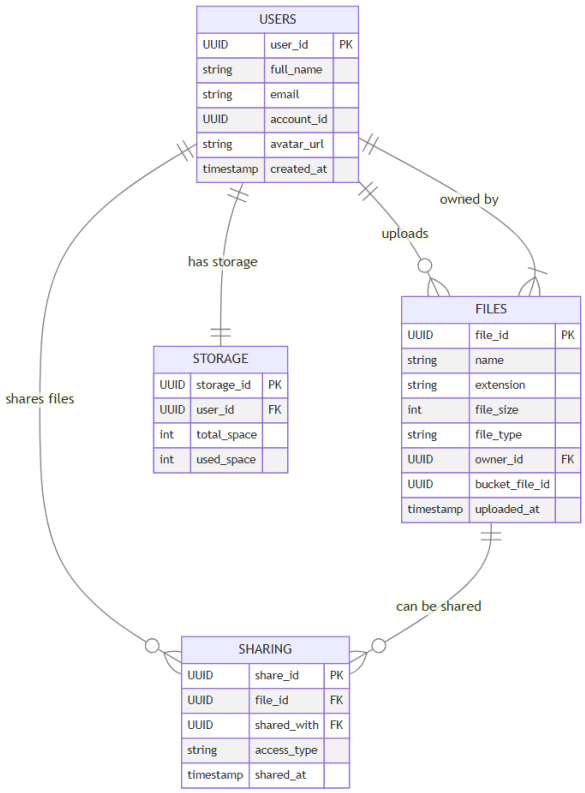
FOREIGN KEY (file_id) REFERENCES files(file_id) ON DELETE CASCADE,

FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE

);

- **Primary Key:** log_id
- **Foreign Keys:**
  - file_id → files(file_id)
  - user_id → users(user_id)
- **Constraints:** action can only be "upload", "rename", "delete", or "share"
- **Relationships:**
  - Tracks user interactions with files

Database Relationships Overview



4.2.3 Security Issues

Authentication & Authorization Risks

- Weak authentication (vulnerable to phishing/email hijacking).
- Risk of session hijacking if cookies are not secured.
- Lack of Multi-Factor Authentication (MFA) increases account compromise risks.

Data Security & Privacy Risks

- Unencrypted data storage exposes sensitive user files.
- Metadata leaks (email, file details) can lead to privacy breaches.
- Weak access control can allow unauthorized file access.

API & Backend Security Risks

- Injection attacks (SQL/NoSQL) due to unsensitized inputs.
- Insecure API endpoints allow unauthorized access.
- No rate limiting makes APIs vulnerable to brute-force & DoS attacks.

File Upload & Sharing Security Risks

- Risk of **malware uploads** if file validation is weak.
- Insufficient file type validation may allow dangerous files.
- Public file links can be misused, leading to data leaks.

Compliance & Legal Risks

- Non-compliance with **GDPR, CCPA, IT Act 2000** can cause legal issues.
- No audit logs make tracking security incidents difficult.
- Weak data retention policies can lead to unauthorized data storage.

CHAPTER NO. 05

IMPLEMENTATION AND TESTING

Implementation is the execution or practice of a plan, method, or design. It involves putting an idea, model, specification, standard, or policy into action. Implementation ensures that the preliminary planning and design materialize into a functional system or product.

Testing is the process of evaluating the functionality of a software program. It identifies errors, gaps, and discrepancies to ensure the application meets the desired expectations before deployment. Testing ensures that the software is reliable, secure, and ready for live use.

5.1 Implementation Approaches

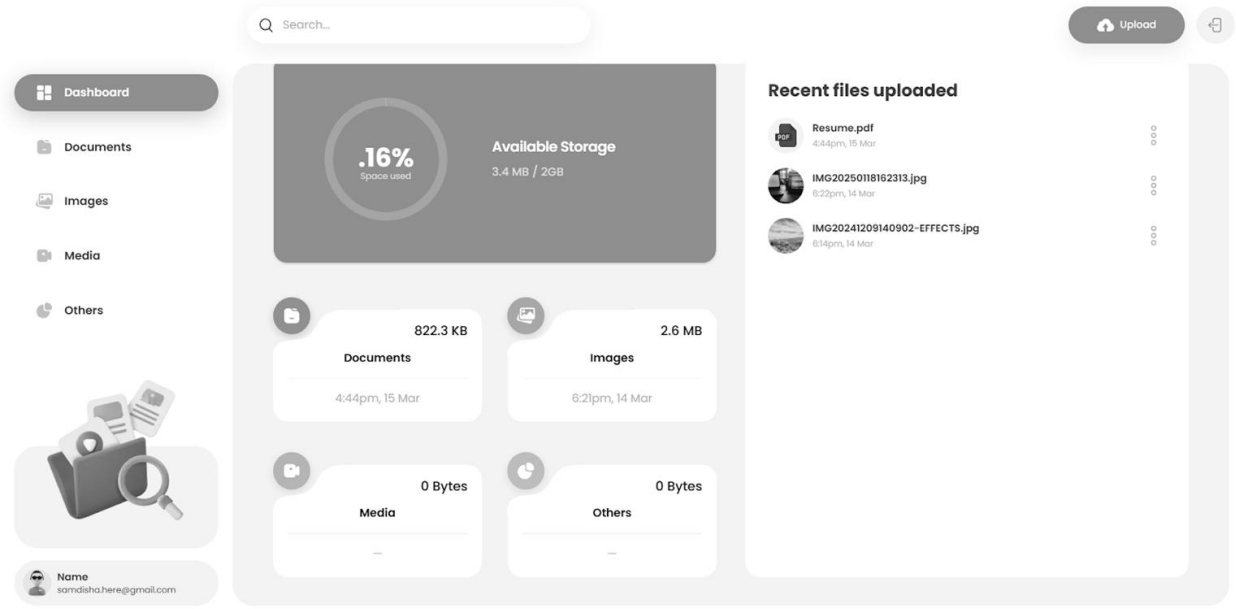
The implementation of the **Storage and File Sharing System** involves a structured approach to ensure scalability, security, and efficiency. The frontend is developed using Next.js 15 and React 19, offering a seamless and responsive user experience with TailwindCSS and ShadCN for modern UI design. The backend is powered by Appwrite, which handles authentication, database management, and cloud storage. Serverless functions are used to automate background tasks like file processing and notifications, ensuring smooth system operations.

A robust database schema is designed using Appwrite Databases to efficiently store user profiles, files, and access permissions. The platform supports multi-file uploads, real-time collaboration, and secure file sharing with role-based access control (RBAC). Users can generate shareable links with expiration, preview supported file types, and manage versions of their documents. To optimize performance, caching mechanisms, lazy loading, and concurrent rendering are implemented, enhancing the system's responsiveness.

Security is a key priority, with end-to-end encryption for sensitive data, secure session management with httpOnly cookies, and rate limiting to prevent brute-force attacks. The platform also integrates real-time notifications to alert users about shared files or changes. By following a systematic implementation strategy, the platform ensures a seamless, secure, and scalable file-sharing experience for users.

The Screens and Description:

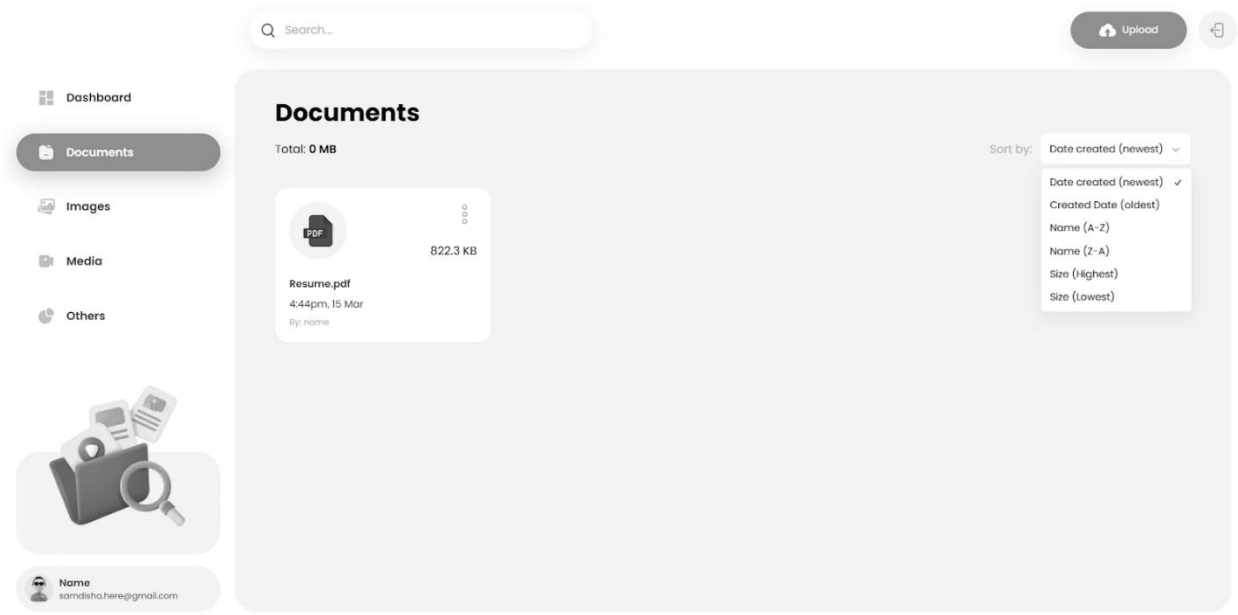
1. The **Dashboard** provides a comprehensive view of the user's storage usage and file management. The central storage indicator visually represents the percentage of space used, along with the total storage capacity (2GB). Below, categorized file sections display the storage consumed by different media types, including Documents, Images, Media, and Others. The recent files uploaded panel on the right shows the latest uploads with timestamps for easy access. Additionally, users can conveniently upload new files using the Upload button at the top right corner, ensuring seamless file management.



It also displays a list of uploaded documents along with essential details such as:

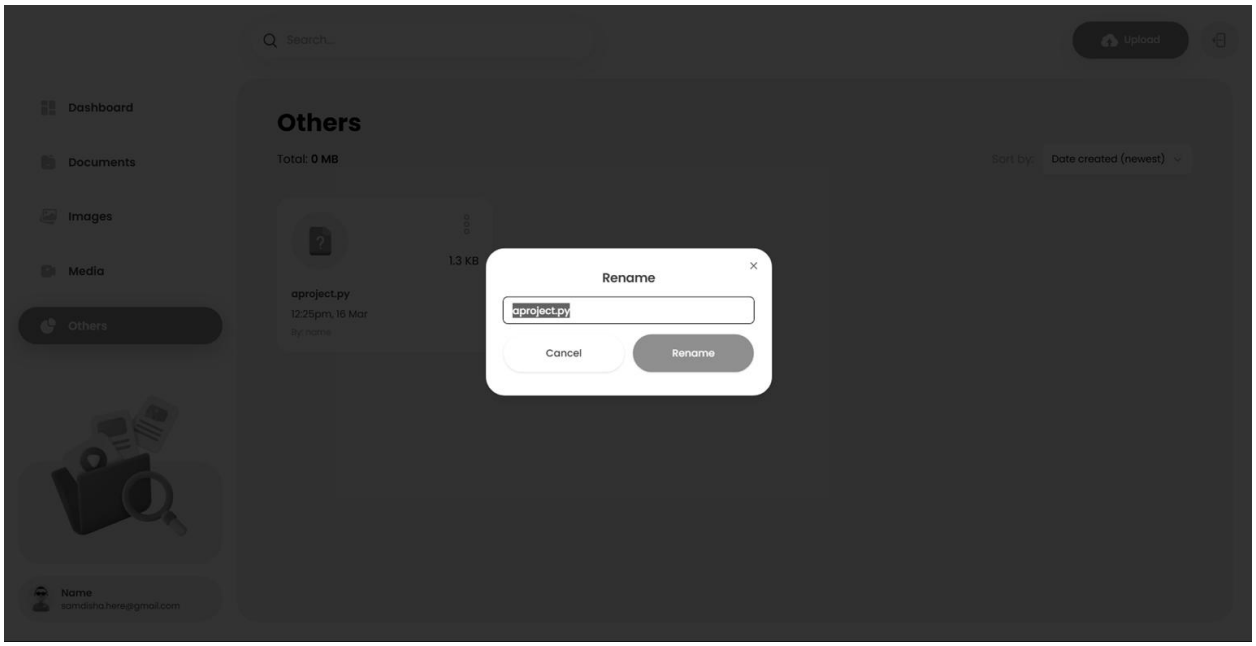
- File Name – The name of the document.
- Upload Time – The date and time the document was uploaded.
- Uploader Information – The name or email of the user who uploaded the file.
- File Size – The storage space occupied by the document.

Additionally, users can sort libraries based on different criteria, such as date created. The Upload button at the top right allows users to add new documents seamlessly. This section ensures smooth organization, retrieval, and sharing of files within the system.

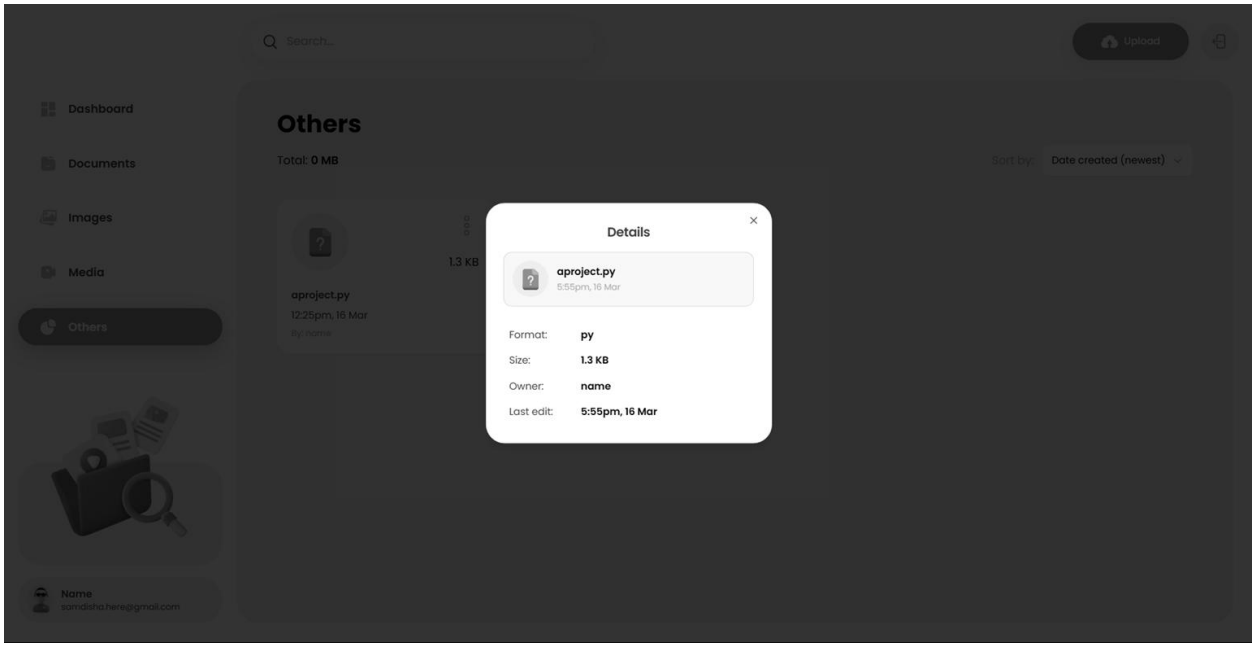


Each item is displayed with details such as name, upload time, and uploader. Users can interact with files through a context menu, which provides the following options:

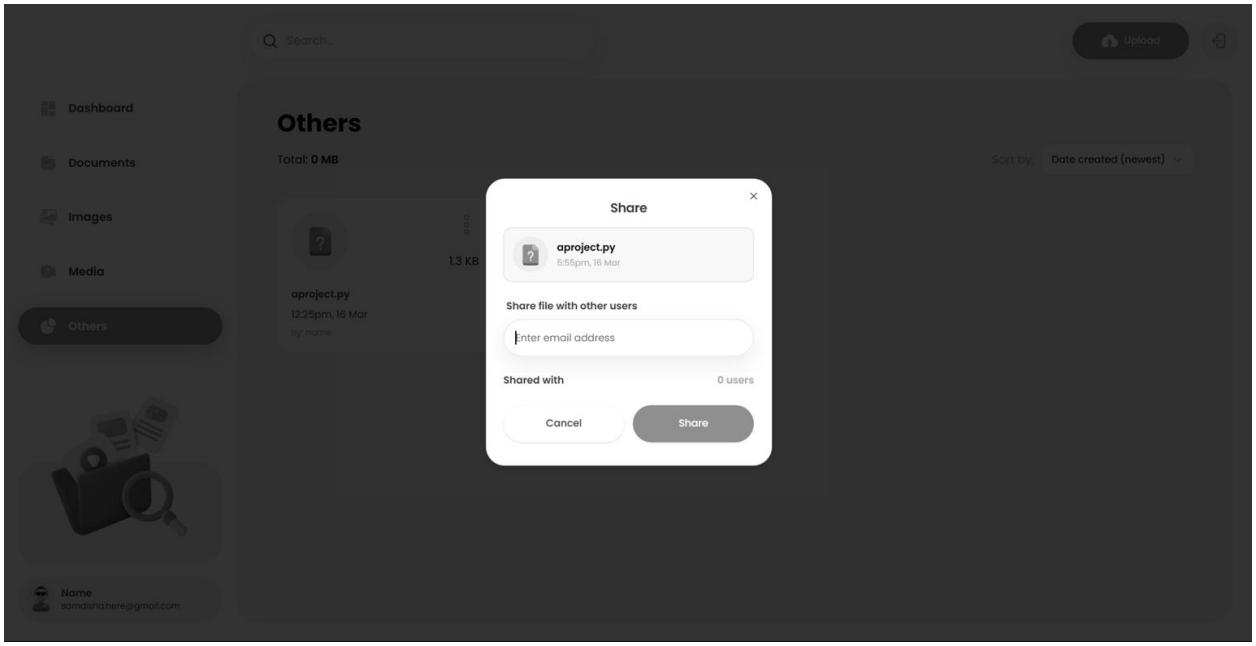
- Rename – Modify the file name.



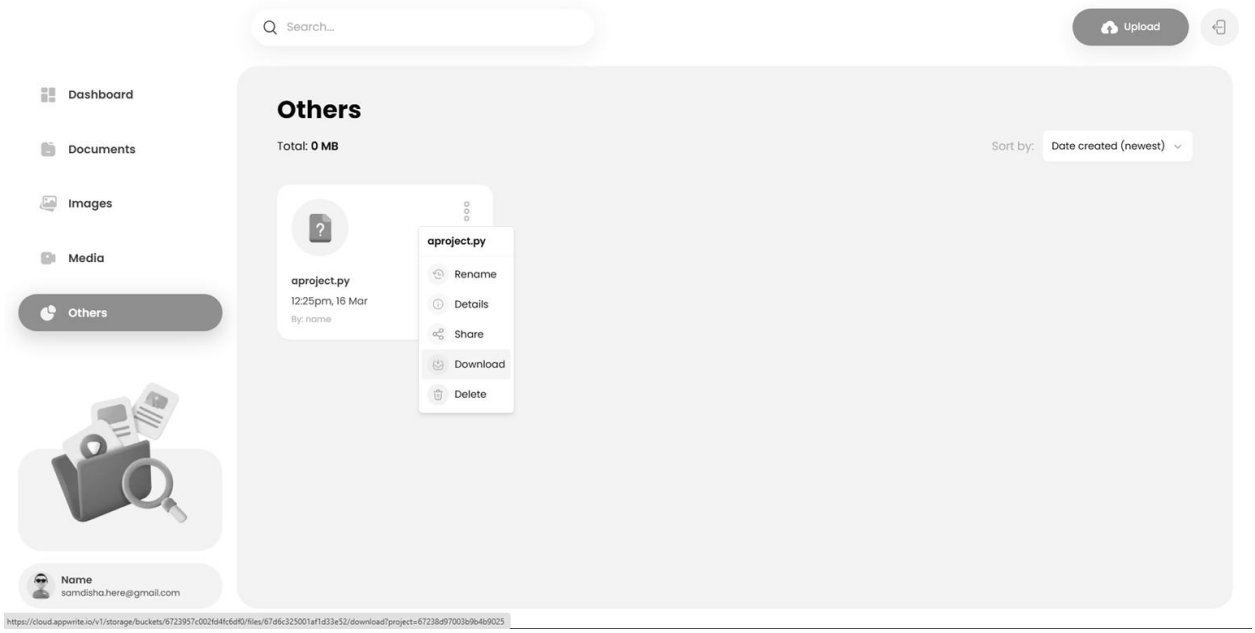
- Details – View additional information about the file.



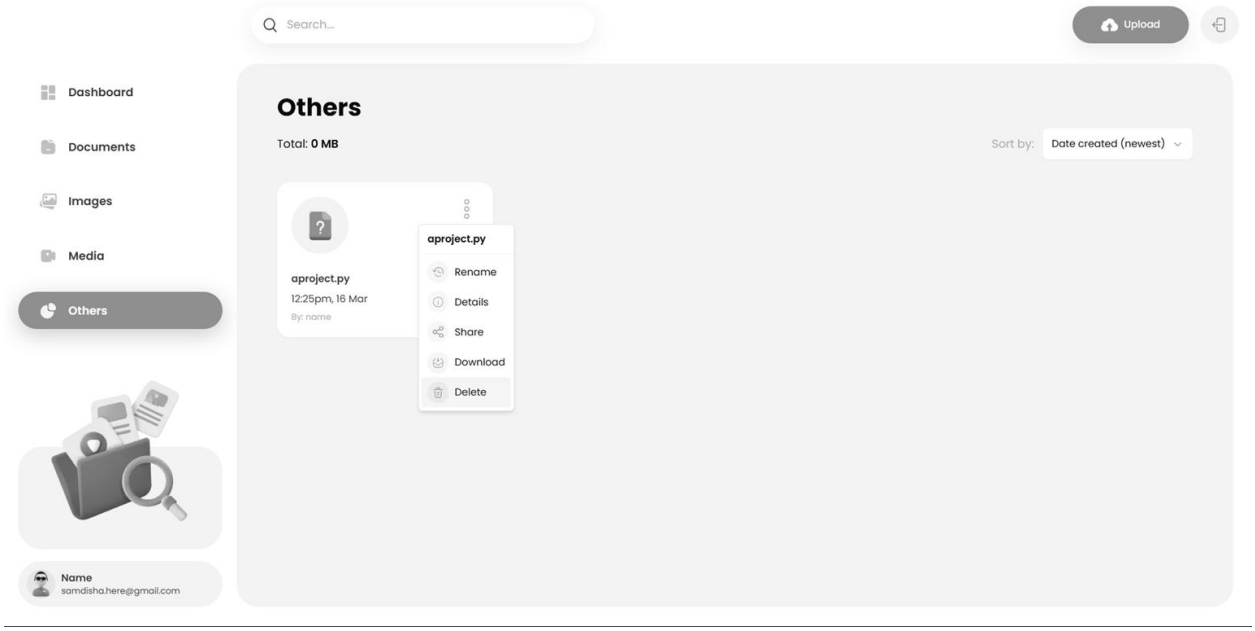
- Share – Share to the receiver's UI using email ID.



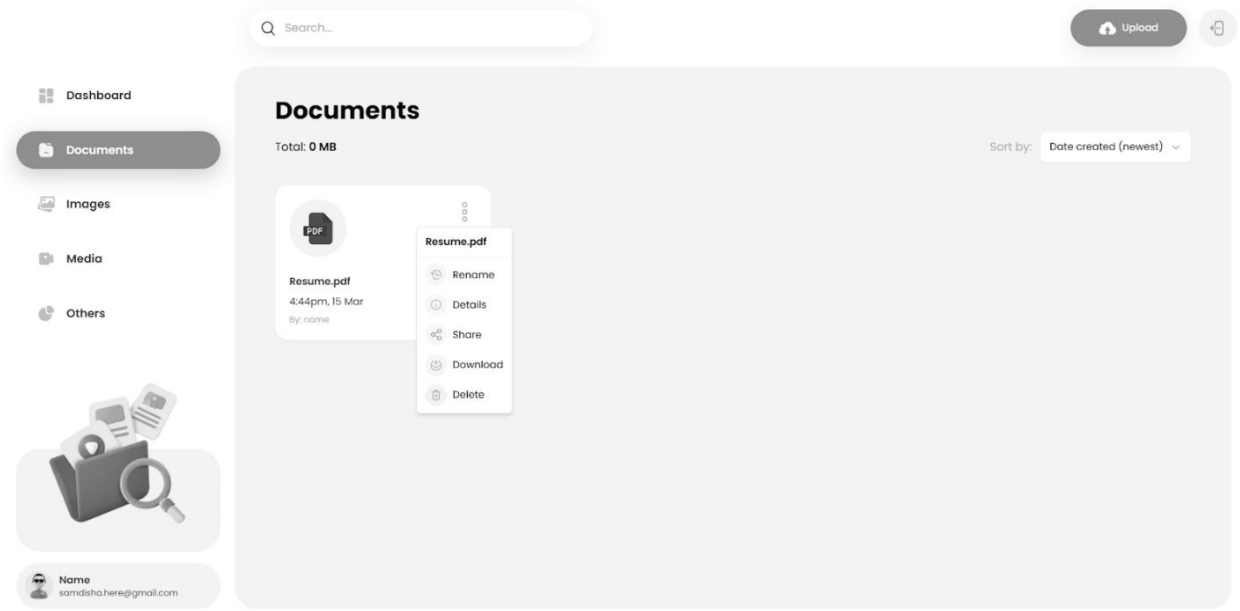
- Download – Save the file to a local device.



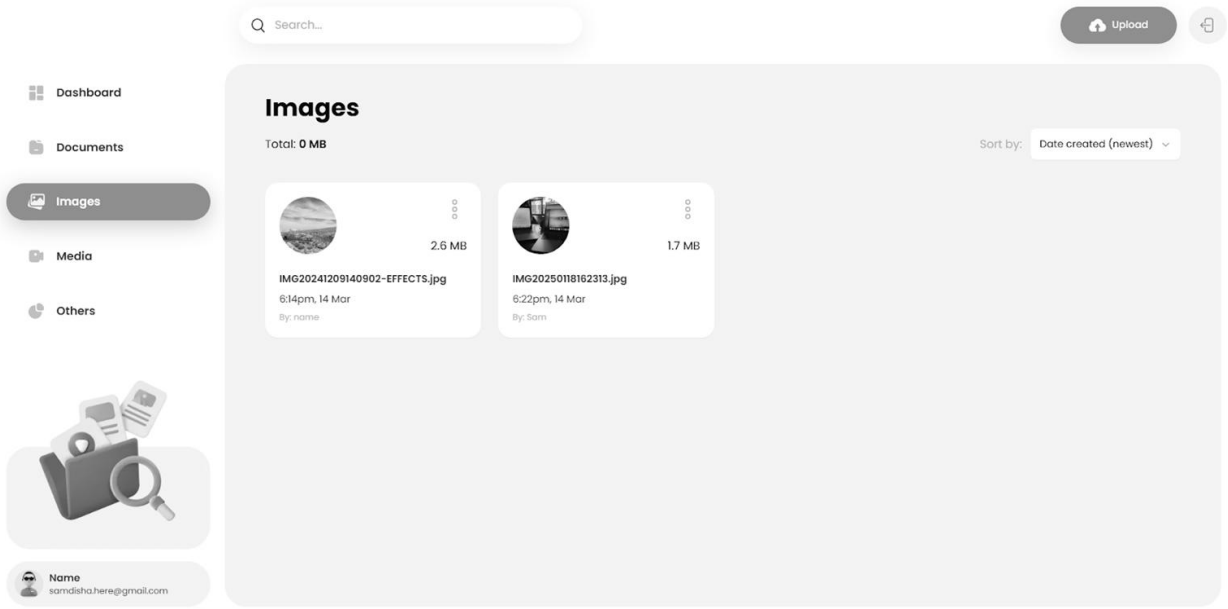
- Delete – Remove the file permanently.



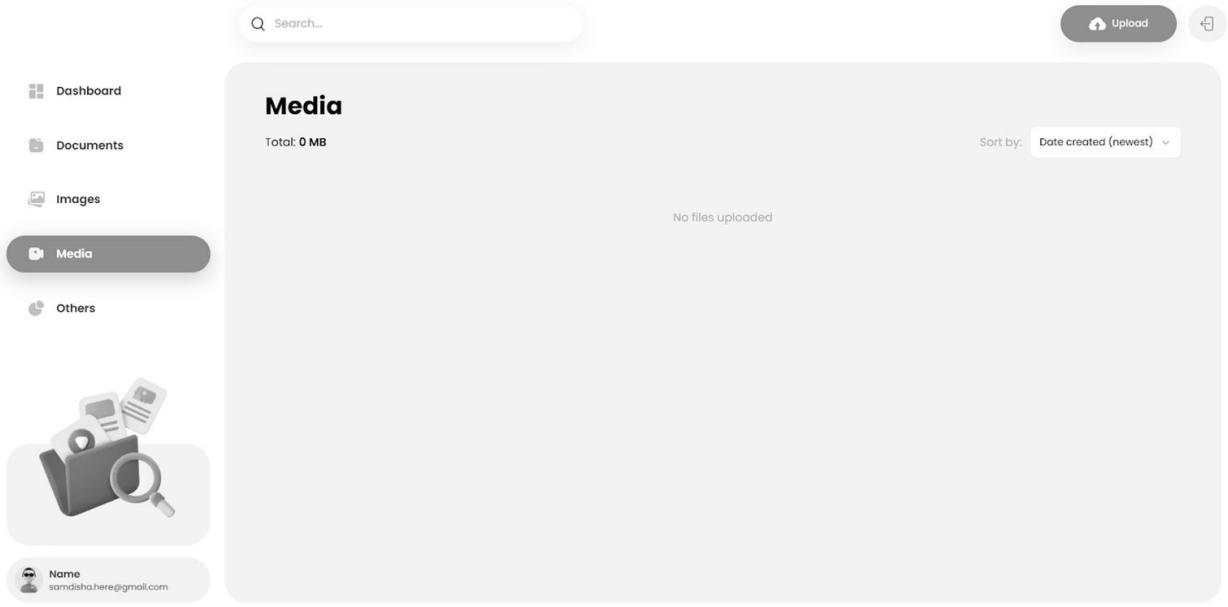
2. The **Documents** section in the storage and file-sharing system provides users with a dedicated space to upload, manage, and organize document files.



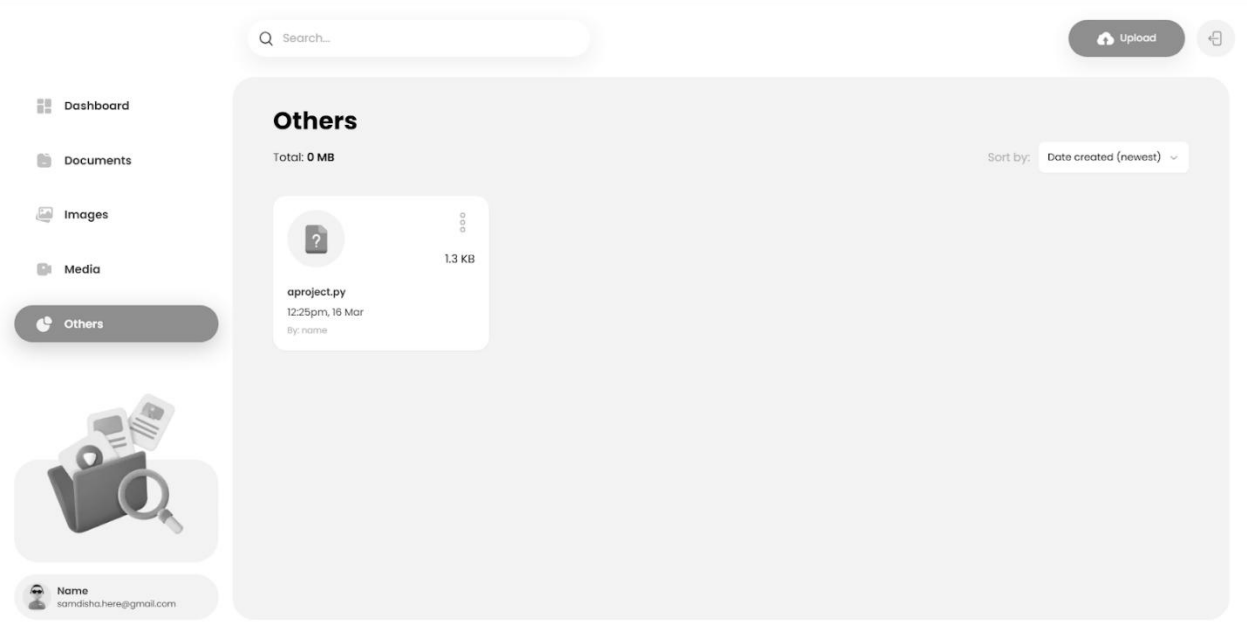
3. The **Images** section in the storage and file-sharing system allows users to upload, view, and manage image files efficiently. It provides a categorized space specifically for image-related content. This section ensures a user-friendly experience for managing image files while allowing easy access, sharing, and organization.



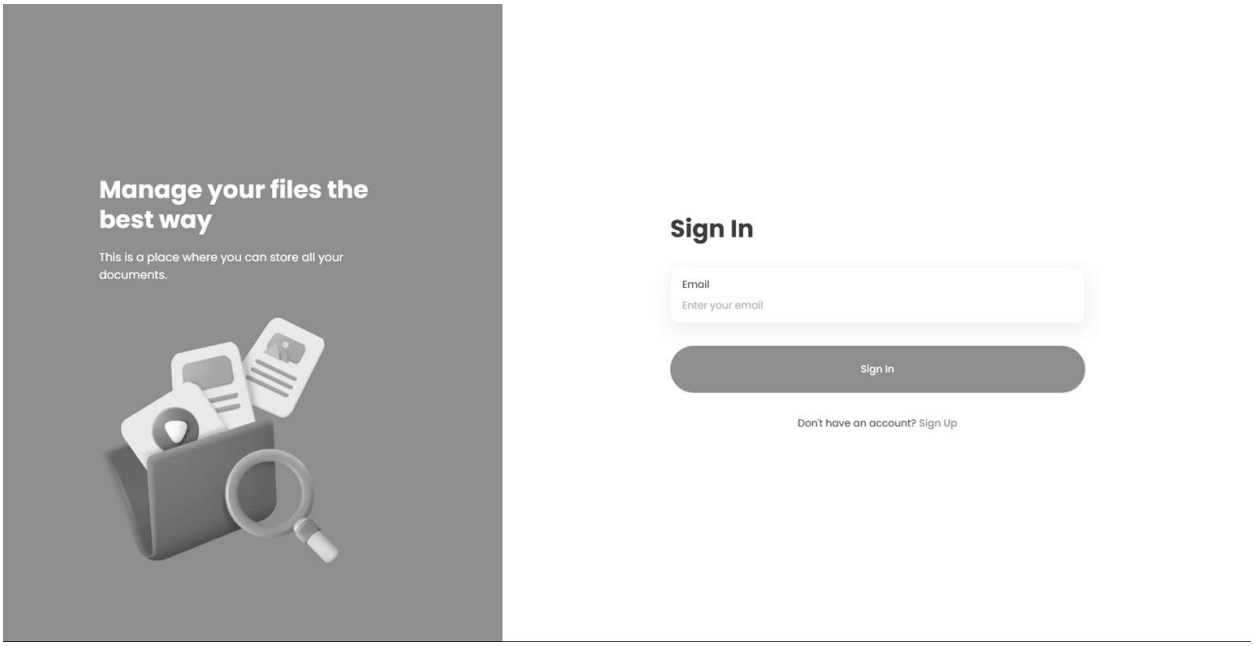
4. The **Media** section in the storage and file-sharing system is dedicated to handling various media files such as audio and video. This section helps users efficiently store, manage, and access their multimedia files. The Media section ensures that users can conveniently store and manage their audio and video files while providing easy access and sharing options.



5. The **Others** section in the storage and file-sharing system is designed to accommodate files that do not fall under the predefined categories such as Documents, Images, or Media. This section allows users to store miscellaneous file types efficiently. The Others section provides a flexible space for users to manage different file types that may not fit into standard categories, ensuring a more comprehensive storage experience.



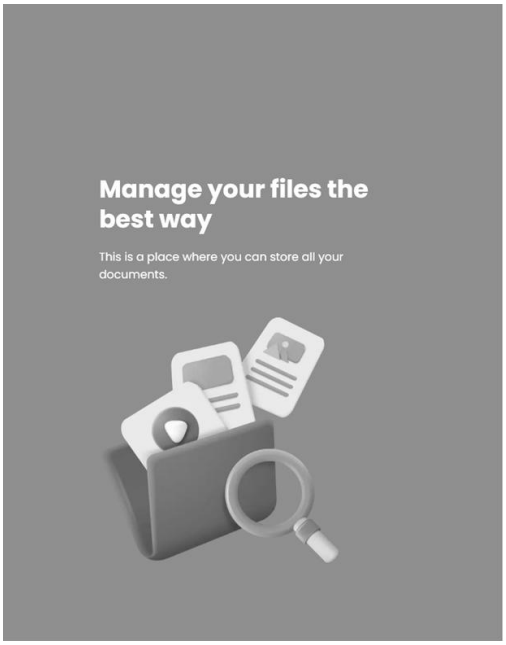
6. The Sign-In Page enables users to securely access their accounts using an OTP-based authentication system. Instead of traditional passwords, users enter their email, receive a One-Time Password (OTP) via email, and use it to log in. This enhances security and simplifies the login process.

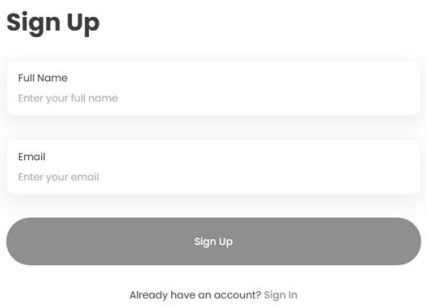


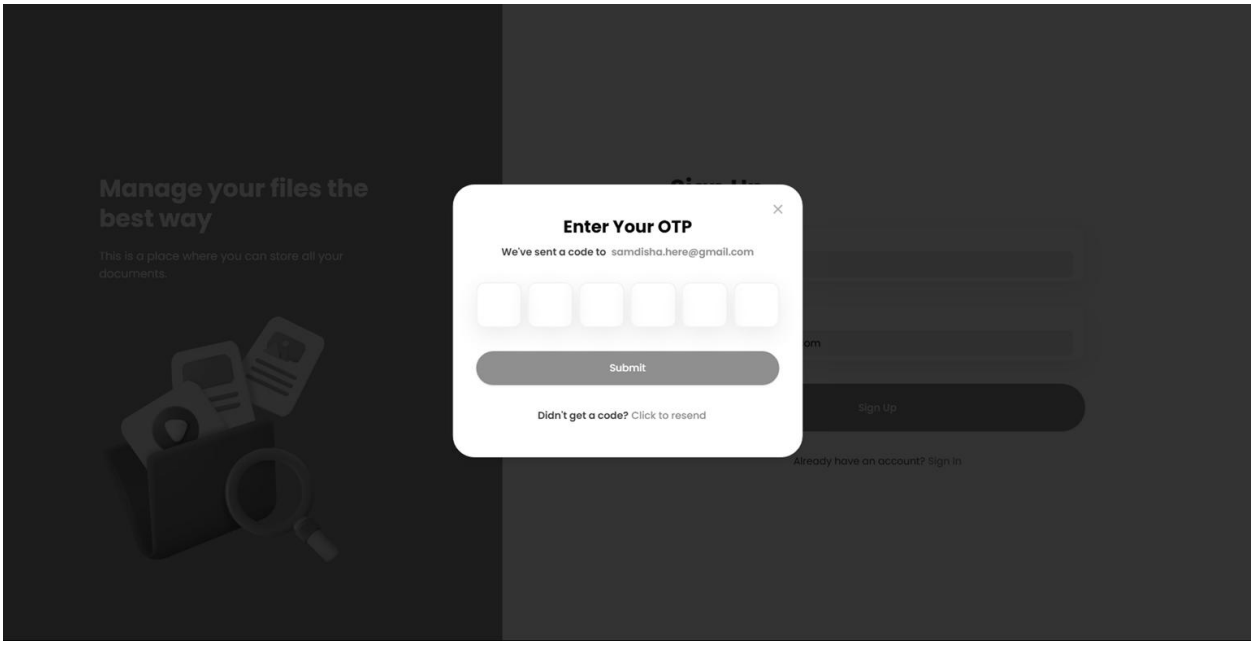
Features

1. OTP-Based Authentication:
    - Users enter their email, and an OTP is sent to their registered email address.
    - Eliminates the need for remembering passwords.
2. Secure and Fast Login:
    - The OTP expires after a short period for security.
    - Ensures a frictionless and secure sign-in process.
3. Responsive Design:
    - Works seamlessly across desktops, tablets, and mobile devices.
4. Error Handling:
    - Displays errors for invalid emails or incorrect OTP entries.
    - Allows resending OTP in case the user does not receive it.

7. The Sign-Up Page allows new users to create an account using a simple and secure OTP-based authentication system. Instead of setting a password, users register using their email address and verify their identity through a One-Time Password (OTP) sent via email. This ensures a streamlined onboarding experience with enhanced security.







5.2 Coding Details and Coding Efficiency

Coding Efficiency

The platform has been designed with an emphasis on performance optimization and maintainability. Key efficiency aspects include:

1. Component-Based Architecture:
    - Reusable React components to reduce redundancy.
    - Efficient state management ensuring smooth updates.
2. Optimized File Handling:
    - File uploads use efficient storage allocation.
    - Lazy loading for images and media files to improve page load speed.
3. Efficient Database Queries:
    - Indexed data retrieval for quick file searches.
    - Optimized queries to minimize unnecessary database calls.
4. Minimalistic and Responsive UI:
    - TailwindCSS ensures lightweight styling.
    - Responsive design for seamless access across devices.
5. Security & Performance Enhancements:
    - Appwrite's built-in security features, such as authentication and role-based access control.
    - Secure file sharing with restricted permissions.
    - Debouncing techniques applied for search functionalities to enhance performance.

By incorporating best practices in software development, the Storage and File Sharing Platform maintains high performance, security, and user-friendly functionality while ensuring scalability for future enhancements.

Coding Details

Sign-in - Page.tsx

```
import AuthForm from "@/components/AuthForm";

const SignIn = () => <AuthForm type="sign-in" />;

export default SignIn;
```

Sign-up - Page.tsx

```
import AuthForm from "@/components/AuthForm";

const SignUp = () => <AuthForm type="sign-up" />;

export default SignUp;
```

Layout.tsx

```
import React from "react";

import Image from "next/image";


const Layout = ({ children }: { children: React.ReactNode }) => {

  return (

    <div className="flex min-h-screen">

      <section className="hidden w-1/2 items-center justify-center bg-brand p-10 lg:flex xl:w-2/5">

        <div className="flex max-h-[800px] max-w-[430px] flex-col justify-center space-y-12">

          <Image

            src="/assets/icons/logo-full.svg"
```

```
          alt="logo"
          width={224}
          height={82}
          className="h-auto"
        />

        <div className="space-y-5 text-white">
          <h1 className="h1">Manage your files the best way</h1>
          <p className="body-1">
            This is a place where you can store all your documents.
          </p>
        </div>
        <Image
          src="/assets/images/files.png"
          alt="Files"
          width={342}
          height={342}
          className="transition-all hover:rotate-2 hover:scale-105"
        />
      </div>
    </section>
```

```
    <section className="flex flex-1 flex-col items-center bg-white p-4 py-10 lg:justify-center lg:p-10 lg:py-0">
      <div className="mb-16 lg:hidden">
       <Image
         src="/assets/icons/logo-full-brand.svg"
         alt="logo"
         width={224}
         height={82}
         className="h-auto w-[200px] lg:w-[250px]"
       />
      </div>

      {children}
    </section>
  </div>
 );
};

export default Layout;
```

Authform.tsx

```
"use client";
```

```
import { z } from "zod";
import { zodResolver } from "@hookform/resolvers/zod";
import { useForm } from "react-hook-form";

import { Button } from "@/components/ui/button";
import {
  Form,
  FormControl,
  FormField,
  FormItem,
  FormLabel,
  FormMessage,
} from "@/components/ui/form";
import { Input } from "@/components/ui/input";
import { useState } from "react";
import Image from "next/image";
import Link from "next/link";
import { createAccount, signInUser } from "@/lib/actions/user.actions";
import OtpModal from "@/components/OTPModal";

type FormType = "sign-in" | "sign-up";

const authFormSchema = (formType: FormType) => {
```

```
  return z.object({
    email: z.string().email(),
    fullName:
      formType === "sign-up"
        ? z.string().min(2).max(50)
        : z.string().optional(),
  });
};

const AuthForm = ({ type }: { type: FormType }) => {
  const [isLoading, setIsLoading] = useState(false);
  const [errorMessage, setErrorMessage] = useState("");
  const [accountId, setAccountId] = useState(null);

  const formSchema = authFormSchema(type);
  const form = useForm<z.infer<typeof formSchema>>({
    resolver: zodResolver(formSchema),
    defaultValues: {
      fullName: "",
      email: "",
    },
  });
```

```
  const onSubmit = async (values: z.infer<typeof formSchema>) => {
    setIsLoading(true);
    setErrorMessage("");

    try {
      const user =
        type === "sign-up"
          ? await createAccount({
              fullName: values.fullName || "",
              email: values.email,
            })
          : await signInUser({ email: values.email });

      setAccountId(user.accountId);
    } catch {
      setErrorMessage("Failed to create account. Please try again.");
    } finally {
      setIsLoading(false);
    }
  };

  return (
    <>
```

```
<Form {...form}>
  <form onSubmit={form.handleSubmit(onSubmit)} className="auth-form">
    <h1 className="form-title">
      {type === "sign-in" ? "Sign In" : "Sign Up"}
    </h1>
    {type === "sign-up" && (
      <FormField
        control={form.control}
        name="fullName"
        render={({ field }) => (
          <FormItem>
            <div className="shad-form-item">
              <FormLabel className="shad-form-label">Full Name</FormLabel>

              <FormControl>
                <Input
                  placeholder="Enter your full name"
                  className="shad-input"
                  {...field}
                />
              </FormControl>
            </div>
```

```
            <FormMessage className="shad-form-message" />
          </FormItem>
        )}
      />
    )}

    <FormField
      control={form.control}
      name="email"
      render={({ field }) => (
        <FormItem>
          <div className="shad-form-item">
            <FormLabel className="shad-form-label">Email</FormLabel>

            <FormControl>
              <Input
                placeholder="Enter your email"
                className="shad-input"
                {...field}
              />
            </FormControl>
          </div>
```

```
            <FormMessage className="shad-form-message" />
          </FormItem>
        )}
      />

      <Button
        type="submit"
        className="form-submit-button"
        disabled={isLoading}
      >
        {type === "sign-in" ? "Sign In" : "Sign Up"}

        {isLoading && (
          <Image
            src="/assets/icons/loader.svg"
            alt="loader"
            width={24}
            height={24}
            className="ml-2 animate-spin"
          />
        )}
      </Button>
```

```
        {errorMessage && <p className="error-message">*{errorMessage}</p>}


        <div className="body-2 flex justify-center">
          <p className="text-light-100">
            {type === "sign-in"
              ? "Don't have an account?"
              : "Already have an account?"}
          </p>
          <Link
            href={type === "sign-in" ? "/sign-up" : "/sign-in"}
            className="ml-1 font-medium text-brand"
          >
            {" "}
            {type === "sign-in" ? "Sign Up" : "Sign In"}
          </Link>
        </div>
      </form>
    </Form>


    {accountId && (
      <OtpModal email={form.getValues("email")} accountId={accountId} />
    )}
  </>
```

```
  );
};

export default AuthForm;
```

Layout.tsx

```
import React from "react";
import Sidebar from "@/components/Sidebar";
import MobileNavigation from "@/components/MobileNavigation";
import Header from "@/components/Header";
import { getCurrentUser } from "@/lib/actions/user.actions";
import { redirect } from "next/navigation";
import { Toaster } from "@/components/ui/toaster";

export const dynamic = "force-dynamic";

const Layout = async ({ children }: { children: React.ReactNode }) => {
  const currentUser = await getCurrentUser();

  if (!currentUser) return redirect("/sign-in");

  return (
    <main className="flex h-screen">
      <Sidebar {...currentUser} />
```

```
      <section className="flex h-full flex-1 flex-col">
        <MobileNavigation {...currentUser} />
        <Header userId={currentUser.$id} accountId={currentUser.accountId} />
        <div className="main-content">{children}</div>
      </section>

      <Toaster />
    </main>
  );
};
export default Layout;
```

Page.tsx

```
import Image from "next/image";
import Link from "next/link";
import { Models } from "node-appwrite";

import ActionDropdown from "@/components/ActionDropdown";
import { Chart } from "@/components/Chart";
import { FormattedDateTime } from "@/components/FormattedDateTime";
import { Thumbnail } from "@/components/Thumbnail";
import { Separator } from "@/components/ui/separator";
```

```
import { getFiles, getTotalSpaceUsed } from "@/lib/actions/file.actions";
import { convertFileSize, getUsageSummary } from "@/lib/utils";


const Dashboard = async () => {
  // Parallel requests
  const [files, totalSpace] = await Promise.all([
    getFiles({ types: [], limit: 10 }),
    getTotalSpaceUsed(),
  ]);


  // Get usage summary
  const usageSummary = getUsageSummary(totalSpace);


  return (
    <div className="dashboard-container">
      <section>
        <Chart used={totalSpace.used} />


        {/* Uploaded file type summaries */}
        <ul className="dashboard-summary-list">
          {usageSummary.map((summary) => (
            <Link
              href={summary.url}
```

```
          key={summary.title}
          className="dashboard-summary-card"
        >
          <div className="space-y-4">
            <div className="flex justify-between gap-3">
              <Image
                src={summary.icon}
                width={100}
                height={100}
                alt="uploaded image"
                className="summary-type-icon"
              />
              <h4 className="summary-type-size">
                {convertFileSize(summary.size) || 0}
              </h4>
            </div>

            <h5 className="summary-type-title">{summary.title}</h5>
            <Separator className="bg-light-400" />
            <FormattedDateTime
              date={summary.latestDate}
              className="text-center"
            />
```

```
            </div>
          </Link>
        ))}
      </ul>
    </section>

    {/* Recent files uploaded */}
    <section className="dashboard-recent-files">
      <h2 className="h3 xl:h2 text-light-100">Recent files uploaded</h2>
      {files.documents.length > 0 ? (
        <ul className="mt-5 flex flex-col gap-5">
          {files.documents.map((file: Models.Document) => (
            <Link
              href={file.url}
              target="_blank"
              className="flex items-center gap-3"
              key={file.$id}
            >
              <Thumbnail
                type={file.type}
                extension={file.extension}
                url={file.url}
              />
```

```
            <div className="recent-file-details">
              <div className="flex flex-col gap-1">
                <p className="recent-file-name">{file.name}</p>
                <FormattedDateTime
                  date={file.$createdAt}
                  className="caption"
                />
              </div>
              <ActionDropdown file={file} />
            </div>
          </Link>
        ))}
      </ul>
    ) : (
      <p className="empty-list">No files uploaded</p>
    )}
  </section>
 </div>
 );
};

export default Dashboard;
```

Globals.css

```
@tailwind base;

@tailwind components;

@tailwind utilities;


@layer base {
  /* Define variables and default styling */
  :root {
    --background: 0 0% 100%;
    --foreground: 142 76% 36%;
    --card: 0 0% 100%;
    --card-foreground: 142 76% 36%;
    --popover: 0 0% 100%;
    --popover-foreground: 142 76% 36%;
    --primary: 142 76% 36%;
    --primary-foreground: 0 0% 100%;
    --secondary: 142 54% 96%;
    --secondary-foreground: 142 76% 36%;
    --muted: 142 54% 96%;
    --muted-foreground: 142 76% 36%;
    --accent: 142 54% 96%;
    --accent-foreground: 142 76% 36%;
    --destructive: 0 84.2% 60.2%;
```

```
    --destructive-foreground: 0 0% 98%;
    --border: 142 13% 91%;
    --input: 142 13% 91%;
    --ring: 142 76% 36%;
    --chart-1: 142 76% 36%;
    --chart-2: 142 54% 96%;
    --chart-3: 142 76% 36%;
    --chart-4: 142 70% 42%;
    --chart-5: 142 80% 32%;
    --radius: 0.5rem;
  }

  .dark {
    --background: 142 19% 20%;
    --foreground: 0 0% 100%;
    --card: 142 19% 20%;
    --card-foreground: 0 0% 100%;
    --popover: 142 19% 20%;
    --popover-foreground: 0 0% 100%;
    --primary: 142 76% 36%;
    --primary-foreground: 0 0% 100%;
    --secondary: 142 54% 96%;
    --secondary-foreground: 142 76% 36%;
```

```
    --muted: 142 54% 96%;
    --muted-foreground: 142 76% 36%;
    --accent: 142 54% 96%;
    --accent-foreground: 142 76% 36%;
    --destructive: 0 62.8% 30.6%;
    --destructive-foreground: 0 0% 98%;
    --border: 142 27.9% 16.9%;
    --input: 142 27.9% 16.9%;
    --ring: 142 76% 36%;
    --chart-1: 142 76% 36%;
    --chart-2: 142 54% 96%;
    --chart-3: 142 76% 36%;
    --chart-4: 142 70% 42%;
    --chart-5: 142 80% 32%;
  }

  * {
    @apply border-border scroll-smooth;
  }

  body {
    @apply bg-background text-foreground min-h-screen;
  }
```

```
}

@layer utilities {
  /* ===== TYPOGRAPHY ===== */
  .h1 {
    @apply text-[34px] leading-[42px] font-bold;
  }
  .h2 {
    @apply text-[24px] leading-[36px] font-bold;
  }
  .h3 {
    @apply text-[20px] leading-[28px] font-semibold;
  }
  .h4 {
    @apply text-[18px] leading-[20px] font-medium;
  }
  .h5 {
    @apply text-[16px] leading-[24px] font-semibold;
  }
  .subtitle-1 {
    @apply text-[16px] leading-[24px] font-medium;
  }
  .subtitle-2 {
```

```
  @apply text-[14px] leading-[20px] font-semibold;
 }
 .body-1 {
  @apply text-[16px] leading-[24px] font-normal;
 }
 .body-2 {
  @apply text-[14px] leading-[20px] font-normal;
 }
 .button {
  @apply text-[14px] leading-[20px] font-medium;
 }
 .caption {
  @apply text-[12px] leading-[16px] font-normal;
 }
 .overline {
  @apply text-[10px] leading-[14px] font-normal;
 }

 /* ===== HELPER CLASSES ===== */
 .container {
  @apply mx-auto max-w-7xl px-5;
 }
 .primary-btn {
```

```
    @apply bg-primary hover:bg-primary/80 transition-all rounded-full button !important;
  }
  .flex-center {
    @apply flex items-center justify-center;
  }

  /* Remove scrollbar */
  .remove-scrollbar::-webkit-scrollbar {
    width: 0px;
    height: 0px;
    border-radius: 0px;
  }
  .remove-scrollbar::-webkit-scrollbar-track {
    background: transparent;
  }
  .remove-scrollbar::-webkit-scrollbar-thumb {
    background: transparent;
    border-radius: 0px;
  }
  .remove-scrollbar::-webkit-scrollbar-thumb:hover {
    background: transparent;
  }
}
```

```
@layer components {
  /* ===== SHADCN OVERRIDES ===== */
  .shad-no-focus {
    @apply outline-none ring-offset-transparent focus:ring-transparent focus:ring-offset-0 focus-visible:outline-none focus-visible:ring-0 focus-visible:ring-transparent focus-visible:ring-offset-0 !important;
  }
  .shad-input {
    @apply border-none shadow-none p-0 shad-no-focus placeholder:text-primary/50 body-2 !important;
  }
  .shad-form-item {
    @apply flex h-[78px] flex-col justify-center rounded-xl border border-primary/20 px-4 shadow-drop-1;
  }
  .shad-form-label {
    @apply text-primary pt-2 body-2 w-full !important;
  }
  .shad-form-message {
    @apply text-red body-2 ml-4 !important;
  }
  .shad-alert-dialog {
```

```
  @apply space-y-4 max-w-[95%] sm:w-fit rounded-xl md:rounded-[30px] px-4 md:px-8 py-10 bg-white outline-none !important;
 }
 .shad-submit-btn {
  @apply bg-primary button hover:bg-primary/80 transition-all rounded-full !important;
 }
 .shad-otp {
  @apply w-full flex gap-1 sm:gap-2 justify-between !important;
 }
 .shad-otp-slot {
  @apply text-[40px] font-medium rounded-xl ring-primary shadow-drop-1 text-primary justify-center flex border-2 border-primary/20 size-12 md:size-16 gap-5 !important;
 }
 .shad-sheet {
  @apply pt-0 !important;
 }
 .shad-sheet button,
 .shad-dialog button {
  @apply focus:ring-0 focus:ring-offset-0 focus-visible:border-none outline-none focus-visible:outline-none focus-visible:ring-transparent focus-visible:ring-offset-0 !important;
 }
 .shad-dropdown-item {
  @apply cursor-pointer !important;
```

```
  }
  .shad-dialog {
    @apply rounded-[26px] w-[90%] max-w-[400px] px-6 py-8 !important;
  }
  .shad-chart-title {
    @apply text-white !important;
  }
  .shad-select-item {
    @apply cursor-pointer !important;
  }
  .shad-active {
    @apply bg-primary text-white shadow-drop-2 !important;
  }

  /* ===== COMPONENT STYLES ===== */

  /* Root Layout */
  .main-content {
    @apply remove-scrollbar h-full flex-1 overflow-auto bg-primary/5 px-5 py-7 sm:mr-7
sm:rounded-[30px] md:mb-7 md:px-9 md:py-10 !important;
  }

  /* Header */
```

```
  .header {
    @apply hidden items-center justify-between gap-5 p-5 sm:flex lg:py-7 xl:gap-10 !important;
  }
  .header-wrapper {
    @apply flex-center min-w-fit gap-4 !important;
  }
  .sign-out-button {
    @apply flex-center h-[52px] min-w-[54px] items-center rounded-full bg-primary/10 p-0 text-primary shadow-none transition-all hover:bg-primary/20 !important;
  }
  .header-user {
    @apply my-3 flex items-center gap-2 rounded-full p-1 text-primary sm:justify-center sm:bg-primary/10 lg:justify-start lg:p-3 !important;
  }
  .header-user-avatar {
    @apply aspect-square w-10 rounded-full object-cover !important;
  }

  /* Mobile Navigation */
  .mobile-header {
    @apply flex h-[60px] justify-between px-5 sm:hidden !important;
  }
  .mobile-nav {
```

```
  @apply h5 flex-1 gap-1 text-primary !important;
 }
 .mobile-nav-list {
  @apply flex flex-1 flex-col gap-4 !important;
 }
 .mobile-nav-item {
  @apply flex text-primary gap-4 w-full justify-start items-center h5 px-6 h-[52px] rounded-full !important;
 }
 .mobile-sign-out-button {
  @apply h5 flex h-[52px] w-full items-center gap-4 rounded-full bg-primary/10 px-6 text-primary shadow-none transition-all hover:bg-primary/20 !important;
 }
 .nav-icon {
  @apply w-6 opacity-50 !important;
 }
 .nav-icon-active {
  @apply opacity-100 !important;
 }

 /* Sidebar */
 .sidebar {
```

```
  @apply remove-scrollbar hidden h-screen w-[90px] flex-col overflow-auto px-5 py-7 sm:flex lg:w-[280px] xl:w-[325px] !important;
 }
 .sidebar-nav {
  @apply h5 mt-9 flex-1 gap-1 text-primary !important;
 }
 .sidebar-nav-item {
  @apply flex text-primary gap-4 rounded-xl lg:w-full justify-center lg:justify-start items-center h5 lg:px-[30px] h-[52px] lg:rounded-full !important;
 }
 .sidebar-user-info {
  @apply mt-4 flex items-center justify-center gap-2 rounded-full bg-primary/10 p-1 text-primary lg:justify-start lg:p-3 !important;
 }
 .sidebar-user-avatar {
  @apply aspect-square w-10 rounded-full object-cover !important;
 }

 /* Search */
 .search {
  @apply relative w-full md:max-w-[480px] !important;
 }
 .search-input-wrapper {
```

```
  @apply flex h-[52px] flex-1 items-center gap-3 rounded-full px-4 shadow-drop-3 !important;
  border-color: #22c55e !important;
 }
 .search-input {
  @apply body-2 shad-no-focus placeholder:body-1 w-full border-none p-0 shadow-none placeholder:text-primary/30 !important;
 }
 .search-result {
  @apply absolute left-0 top-16 z-50 flex w-full flex-col gap-3 rounded-[20px] bg-white p-4 !important;
 }
 .empty-result {
  @apply body-2 text-center text-primary !important;
 }

 /* Dashboard */
 .dashboard-container {
  @apply mx-auto grid max-w-7xl grid-cols-1 gap-6 md:grid-cols-2 xl:gap-10 !important;
 }
 .dashboard-summary-list {
  @apply mt-6 grid grid-cols-1 gap-4 xl:mt-10 xl:grid-cols-2 xl:gap-9 !important;
 }
 .dashboard-summary-card {
```

```
    @apply relative mt-6 rounded-[20px] bg-white p-5 transition-all hover:scale-105 !important;
  }
  .summary-type-icon {
    @apply absolute -left-3 top-[-25px] z-10 w-[190px] object-contain !important;
  }
  .summary-type-size {
    @apply h4 relative z-20 w-full text-right !important;
  }
  .summary-type-title {
    @apply h5 relative z-20 text-center !important;
  }
  .dashboard-recent-files {
    @apply h-full rounded-[20px] bg-white p-5 xl:p-8 !important;
  }
  .recent-file-details {
    @apply flex w-full flex-col xl:flex-row xl:justify-between !important;
  }
  .recent-file-name {
    @apply subtitle-2 line-clamp-1 w-full text-primary sm:max-w-[200px] lg:max-w-[250px]
!important;
  }
  .recent-file-date {
    @apply body-2 text-primary/60 !important;
```

```
  }
  .empty-list {
    @apply body-1 mt-10 text-center text-primary/50 !important;
  }

  /* Chart */
  .chart {
    @apply flex items-center rounded-[20px] bg-primary p-5 text-white md:flex-col xl:flex-row !important;
  }
  .chart-container {
    @apply mx-auto aspect-square w-[180px] text-white xl:w-[250px] !important;
  }
  .polar-grid {
    @apply first:fill-white/20 last:fill-primary !important;
  }
  .chart-details {
    @apply flex-1 items-start px-3 py-0 sm:px-5 lg:p-3 xl:pr-5 !important;
  }
  .chart-total-percentage {
    @apply fill-white text-4xl font-bold !important;
  }
  .chart-title {
```

```
  @apply h3 font-bold md:text-center lg:text-left !important;
 }
 .chart-description {
  @apply subtitle-1 mt-2 w-full text-white/70 md:text-center lg:text-left !important;
 }
 .recharts-responsive-container {
  height: initial !important;
 }

 /* Type page */
 .page-container {
  @apply mx-auto flex w-full max-w-7xl flex-col items-center gap-8 !important;
 }
 .total-size-section {
  @apply flex mt-2 flex-col justify-between sm:flex-row sm:items-center !important;
 }
 .file-list {
  @apply grid w-full gap-6 sm:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4 !important;
 }
 .sort-container {
  @apply mt-5 flex items-center sm:mt-0 sm:gap-3 !important;
 }
```

```
  /* Sort */
  .sort-select {
    @apply shad-no-focus h-11 w-full rounded-[8px] border-transparent bg-white !shadow-sm sm:w-[210px] !important;
  }
  .sort-select-content {
    @apply !shadow-drop-3 !important;
  }

  /* Card */
  .file-card {
    @apply flex cursor-pointer flex-col gap-6 rounded-[18px] bg-white p-5 shadow-sm transition-all hover:shadow-drop-3 !important;
  }
  .file-card-details {
    @apply flex flex-col gap-2 text-primary !important;
  }

  /* Thumbnail */
  .thumbnail {
    @apply flex-center size-[50px] min-w-[50px] overflow-hidden rounded-full bg-primary/10;
  }
  .thumbnail-image {
```

```
  @apply size-full object-cover object-center !important;
 }

 /* File Uploader */
 .uploader-button {
  @apply primary-btn h-[52px] gap-2 px-10 shadow-drop-1 !important;
  background-color: #22c55e !important;
 }
 .uploader-preview-list {
  @apply fixed bottom-10 right-10 z-50 flex size-full h-fit max-w-[480px] flex-col gap-3 rounded-[20px] bg-white p-7 shadow-drop-3 !important;
 }
 .uploader-preview-item {
  @apply flex items-center justify-between gap-3 rounded-xl p-3 shadow-drop-3 !important;
 }
 .preview-item-name {
  @apply subtitle-2 mb-2 line-clamp-1 max-w-[300px] !important;
 }

 /* OTP Modal */
 .otp-close-button {
  @apply absolute -right-1 -top-7 cursor-pointer sm:-right-2 sm:-top-4 !important;
 }
```

```
  /* Auth Form */
  .auth-form {
    @apply flex max-h-[800px] w-full max-w-[580px] flex-col justify-center space-y-6 transition-all lg:h-full lg:space-y-8 !important;
  }
  .form-title {
    @apply h1 text-center text-primary md:text-left !important;
  }
  .form-submit-button {
    @apply primary-btn h-[66px] !important;
  }
  .error-message {
    @apply body-2 mx-auto w-fit rounded-xl bg-error/5 px-8 py-4 text-center text-error !important;
  }

  /* Actions Dropdown */
  .rename-input-field {
    @apply body-2 shad-no-focus h-[52px] w-full rounded-full border px-4 shadow-drop-1 !important;
  }
  .delete-confirmation {
    @apply text-center text-primary !important;
```

```
  }
  .delete-file-name {
    @apply font-medium text-primary !important;
  }
  .modal-cancel-button {
    @apply h-[52px] flex-1 rounded-full bg-white text-primary hover:bg-transparent !important;
  }
  .modal-submit-button {
    @apply primary-btn !mx-0 h-[52px] w-full flex-1 !important;
  }

  /* Actions Modal Content */
  .file-details-thumbnail {
    @apply !mb-1 flex items-center gap-3 rounded-xl border border-primary/10 bg-primary/5 p-3
!important;
  }
  .file-details-label {
    @apply body-2 w-[30%] text-primary !important;
  }
  .file-details-value {
    @apply subtitle-2 flex-1 !important;
  }
  .share-wrapper {
```

```
    @apply !mt-2 space-y-2 !important;
  }
  .share-input-field {
    @apply body-2 shad-no-focus h-[52px] w-full rounded-full border px-4 shadow-drop-1 !important;
  }
  .share-remove-user {
    @apply rounded-full bg-transparent text-primary shadow-none hover:bg-transparent !important;
  }
  .remove-icon {
    @apply aspect-square rounded-full !important;
  }

  /* Toast */
  .error-toast {
    @apply bg-red !rounded-[10px] !important;
  }
}

/* Specific color overrides for green theme */
.logo-wrapper svg path,
.logo-wrapper svg circle {
```

```
  fill: var(--primary);
}

/* StorIt logo specific override */
#storeit-logo .main-logo-color {
  fill: #22c55e !important;
}

/* Convert any remaining pink/coral elements to green */
button[class*="bg-\\[\\#ff6b6b\\]"],
[class*="bg-\\[\\#ff6b6b\\]"],
[class*="bg-\\[\\#ff7979\\]"],
[class*="bg-\\[\\#ff8787\\]"] {
  background-color: #22c55e !important;
  color: white !important;
}

button[class*="text-\\[\\#ff6b6b\\]"],
[class*="text-\\[\\#ff6b6b\\]"],
[class*="text-\\[\\#ff7979\\]"],
[class*="text-\\[\\#ff8787\\]"] {
  color: #22c55e !important;
}
```

```
/* Dashboard and chart element overrides */
[class*="dashboard-summary-card"][class*="bg-\\[\\#ff6b6b\\]"],
[class*="dashboard-summary-card"][class*="bg-\\[\\#ff7979\\]"],
[class*="chart"][class*="bg-\\[\\#ff6b6b\\]"],
[class*="chart"][class*="bg-\\[\\#ff7979\\]"] {
  background-color: #22c55e !important;
}

/* SVG fill overrides */
svg[fill="#ff6b6b"],
svg[fill="#ff7979"],
svg[fill="#ff8787"] {
  fill: #22c55e !important;
}

svg path[fill="#ff6b6b"],
svg path[fill="#ff7979"],
svg path[fill="#ff8787"] {
  fill: #22c55e !important;
}

/* Convert hover states */
```

```
[class*="hover\\:bg-\\[\\#ff7979\\]"]:hover,
[class*="hover\\:bg-\\[\\#ff6b6b\\]"]:hover {
  background-color: #4ade80 !important;
}

/* Dashboard ring chart conversion */
.recharts-polar-grid-concentric circle {
  stroke: rgba(34, 197, 94, 0.2) !important;
}

.recharts-polar-grid-concentric circle:last-child {
  stroke: #22c55e !important;
}
```

Main - Layout.tsx

```
import type { Metadata } from "next";
import { Poppins } from 'next/font/google'

import "./globals.css";

const poppins = Poppins({
    subsets: ['latin'],
    weight: ['100', '200', '300', '400', '500', '600', '700', '800', '900'],
```

```
    variable: '--font-poppins',
})

export const metadata: Metadata = {
  title: "StoreIt",
  description: "StoreIt - The only storage solution you need.",
};

export default function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode;
}>) {
  return (
    <html lang="en">
      <body
        className={`${poppins.variable} font-poppins antialiased`}
      >
        {children}
      </body>
    </html>
  );
}
```

Constants - index.ts

```
export const navItems = [
  {
    name: "Dashboard",
    icon: "/assets/icons/dashboard.svg",
    url: "/",
  },
  {
    name: "Documents",
    icon: "/assets/icons/documents.svg",
    url: "/documents",
  },
  {
    name: "Images",
    icon: "/assets/icons/images.svg",
    url: "/images",
  },
  {
    name: "Media",
    icon: "/assets/icons/video.svg",
    url: "/media",
  },
```

```
  {
    name: "Others",
    icon: "/assets/icons/others.svg",
    url: "/others",
  },
];


export const actionsDropdownItems = [
  {
    label: "Rename",
    icon: "/assets/icons/edit.svg",
    value: "rename",
  },
  {
    label: "Details",
    icon: "/assets/icons/info.svg",
    value: "details",
  },
  {
    label: "Share",
    icon: "/assets/icons/share.svg",
    value: "share",
  },
```

```
  {
    label: "Download",
    icon: "/assets/icons/download.svg",
    value: "download",
  },
  {
    label: "Delete",
    icon: "/assets/icons/delete.svg",
    value: "delete",
  },
];

export const sortTypes = [
  {
    label: "Date created (newest)",
    value: "$createdAt-desc",
  },
  {
    label: "Created Date (oldest)",
    value: "$createdAt-asc",
  },
  {
    label: "Name (A-Z)",
```

```
    value: "name-asc",
  },
  {
    label: "Name (Z-A)",
    value: "name-desc",
  },
  {
    label: "Size (Highest)",
    value: "size-desc",
  },
  {
    label: "Size (Lowest)",
    value: "size-asc",
  },
];

export const avatarPlaceholderUrl =
  "https://img.freepik.com/free-psd/3d-illustration-person-with-sunglasses_23-2149436188.jpg";

export const MAX_FILE_SIZE = 50 * 1024 * 1024;
```

UI COMPONENTS

Component Name	Purpose	Category
alert-dialog.tsx	Displays alert modals for user confirmations and warnings.	Dialogs & Alerts
button.tsx	Reusable button component with different styles and actions.	Controls & Inputs
card.tsx	Used to display grouped information inside a card layout.	Layout & Display
chart.tsx	Renders graphical charts for data visualization.	Data & Visualization
dialog.tsx	General modal dialog component for various interactions.	Dialogs & Alerts
dropdown-menu.tsx	Provides a dropdown menu for selecting options.	Navigation & Menus
form.tsx	Handles form layouts and validation.	Forms & Inputs
input-otp.tsx	OTP input field for authentication flows.	Forms & Inputs
input.tsx	Standard text input component.	Forms & Inputs
label.tsx	Label component for form elements.	Forms & Inputs
select.tsx	Custom select dropdown for choosing options.	Forms & Inputs
separator.tsx	UI separator for dividing sections.	Layout & Display
sheet.tsx	Side panel for additional content or actions.	Layout & Display
toast.tsx	Notification component for displaying messages.	Feedback & Alerts

Component Name	Purpose	Category
toaster.tsx	Manages and displays multiple toasts.	Feedback & Alerts
ActionDropdown.tsx	Dropdown component for file actions.	Controls & Inputs
ActionsModalContent.tsx	Renders content inside an action modal.	Dialogs & Alerts
AuthForm.tsx	Authentication form used for sign-in/sign-up.	Forms & Authentication
Card.tsx	Another card component, likely with extended features.	Layout & Display
Chart.tsx	Another chart component, possibly for different types of graphs.	Data & Visualization
FileUploader.tsx	Uploads and manages files in the platform.	Forms & Inputs
FormattedDateTime.tsx	Displays formatted date and time values.	Data Display
Header.tsx	Top navigation header component.	Navigation & Menus
MobileNavigation.tsx	Navigation menu for mobile devices.	Navigation & Menus
OTPModal.tsx	Modal for OTP verification during login/signup.	Dialogs & Alerts
Search.tsx	Search bar component for finding files or data.	Navigation & Menus
Sidebar.tsx	Sidebar navigation for accessing different sections.	Navigation & Menus
Sort.tsx	Sorting control for lists and tables.	Controls & Inputs
Thumbnail.tsx	Displays image/file thumbnails.	Data Display

5.3 Testing Approaches

Testing is a critical phase in the development of the **Storage and File Sharing Platform** to ensure reliability, security, and performance. The goal of testing is to identify errors and defects before deployment, guaranteeing a seamless user experience. Various testing methodologies have been employed to validate different system components and interactions, considering the project's tech stack, including Next.js 15, React 19, Appwrite, and TypeScript.

5.3.1 Unit Testing

Unit testing is conducted to verify that individual components of the system function correctly in isolation. This testing is automated and executed during the development phase to detect early-stage defects.

Test Objectives:

- Ensure that all form fields, buttons, and file management functions work correctly.
- Validate the proper format of input fields (e.g., authentication, file names).
- Prevent duplicate entries in the storage system.
- Verify responsiveness of UI elements across various devices.

Features to be Tested:

- Correct format validation of all input fields.
- Proper activation of buttons and form navigation.
- File upload validation for type, size, and format.
- Secure authentication and session management.

5.3.2 Integration Testing

Integration testing is performed to ensure that different system modules interact correctly. It checks communication between components such as user authentication, file upload, sharing, and permission handling.

Objectives:

- Validate seamless integration between the **frontend (Next.js 15) and backend (Appwrite services)**.
- Ensure smooth communication between Appwrite authentication and storage services.
- Check database operations for consistency and correctness.

5.3.3 User Acceptance Testing (UAT): User Acceptance Testing (UAT) ensures that the system meets end-user requirements and is intuitive to use.

Objectives:

- Validate real-world usability and business requirements.
- Identify UI/UX issues based on user feedback.
- Ensure smooth navigation and interaction.

5.3.4 Performance and Security Testing: This testing ensures that the system is secure and can handle various loads without failure.

Objectives:

- Identify system bottlenecks and optimize performance.
- Test for vulnerabilities such as SQL injection and XSS.
- Ensure that the system handles multiple concurrent users.

By employing unit testing, integration testing, user acceptance testing, and performance & security testing, the Storage and File Sharing Platform ensures a robust, secure, and efficient user experience before deployment.

5.3.5 Test Cases

Unit Test Cases:

Sr. No.	Module	Test Case	Expected Result
1	User Authentication	Validate login with valid and invalid credentials	Proper access control
2	Form Validation	Ensure all input fields accept correct formats	Only valid data is accepted
3	File Upload	Upload a file exceeding the size limit	System prevents upload and shows an error
4	File Upload	Upload a file with an unsupported format	System rejects file and displays a warning
5	UI Responsiveness	Test interface on different screen sizes	UI adapts properly across devices

Integration Test Cases:

Sr. No.	Module	Test Case	Expected Result
1	Authentication	Login triggers Appwrite authentication flow	User successfully logs in or gets an error
2	File Upload	Uploaded file is stored in the correct database	File appears in Appwrite storage
3	File Sharing	Shared file is accessible by permitted users	Only authorized users can access the file
4	Notifications	Notify user when a file is shared with them	User receives a notification

UAT Test Cases:

Sr. No.	Module	Test Case	Expected Result
1	Registration	New user successfully creates an account	User account is created successfully
2	Dashboard	User navigates to dashboard	Dashboard displays correct data
3	File Download	User downloads a shared file	File is successfully downloaded

Performance & Security Test Cases:

Sr. No.	Module	Test Case	Expected Result
1	Performance	Simulate 100 concurrent users uploading files	System remains stable with no crashes
2	Security	Attempt injection	System prevents unauthorized access
3	Security	Test session expiration after logout	User session is properly terminated
4	Performance	Upload 100 large files at once	System manages uploads efficiently

CHAPTER NO. 06

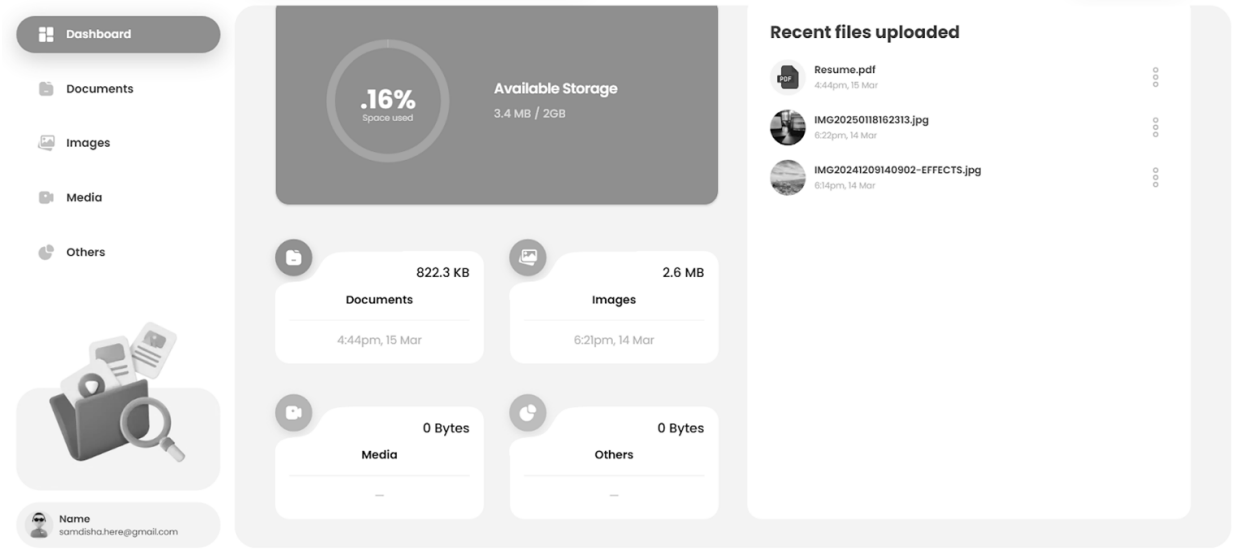
RESULTS AND DISCUSSION

6.1 Test Report

Description: The **Storage and File Sharing Platform** provides real-time storage usage reports, offering users an intuitive way to monitor their file consumption. The dashboard visually represents the total storage used out of the allocated **2GB**, displaying a percentage-based usage metric.

The report categorizes files into different media types such as **Documents, Images, Media, and Others**, allowing users to track how much space each category occupies. Recent file uploads are also displayed, showing their timestamps for easy reference.

Users can leverage this real-time reporting system to efficiently manage their storage, delete unnecessary files, and ensure optimal usage of the available space.




```
import Image from "next/image";

import Link from "next/link";

import { Models } from "node-appwrite";
```

```
import ActionDropdown from "@/components/ActionDropdown";
import { Chart } from "@/components/Chart";
import { FormattedDateTime } from "@/components/FormattedDateTime";
import { Thumbnail } from "@/components/Thumbnail";
import { Separator } from "@/components/ui/separator";
import { getFiles, getTotalSpaceUsed } from "@/lib/actions/file.actions";
import { convertFileSize, getUsageSummary } from "@/lib/utils";
const Dashboard = async () => {
  // Parallel requests
  const [files, totalSpace] = await Promise.all([
    getFiles({ types: [], limit: 10 }),
    getTotalSpaceUsed(),
  ]);
  // Get usage summary
  const usageSummary = getUsageSummary(totalSpace);
  return (
    <div className="dashboard-container">
      <section>
        <Chart used={totalSpace.used} />

        {/* Uploaded file type summaries */}
        <ul className="dashboard-summary-list">
```

```
{usageSummary.map((summary) => (
 <Link
  href={summary.url}
  key={summary.title}
  className="dashboard-summary-card"
 >
  <div className="space-y-4">
   <div className="flex justify-between gap-3">
    <Image
     src={summary.icon}
     width={100}
     height={100}
     alt="uploaded image"
     className="summary-type-icon"
    />
    <h4 className="summary-type-size">
     {convertFileSize(summary.size) || 0}
    </h4>
   </div>

   <h5 className="summary-type-title">{summary.title}</h5>
   <Separator className="bg-light-400" />
   <FormattedDateTime
```

```
              date={summary.latestDate}
              className="text-center"
            />
          </div>
        </Link>
      ))}
    </ul>
  </section>
  {/* Recent files uploaded */}
  <section className="dashboard-recent-files">
    <h2 className="h3 xl:h2 text-light-100">Recent files uploaded</h2>
    {files.documents.length > 0 ? (
      <ul className="mt-5 flex flex-col gap-5">
        {files.documents.map((file: Models.Document) => (
          <Link
            href={file.url}
            target="_blank"
            className="flex items-center gap-3"
            key={file.$id}
          >
            <Thumbnail
              type={file.type}
              extension={file.extension}
```

```
              url={file.url}
            />
            <div className="recent-file-details">
              <div className="flex flex-col gap-1">
                <p className="recent-file-name">{file.name}</p>
                <FormattedDateTime
                  date={file.$createdAt}
                  className="caption"
                />
              </div>
              <ActionDropdown file={file} />
            </div>
          </Link>
        ))}
      </ul>
    ) : (
      <p className="empty-list">No files uploaded</p>
    )}
  </section>
 </div>
 );
};
export default Dashboard;
```

6.2 User Documentation

1. Introduction

The Storage and File Sharing Platform is a powerful tool designed to help users upload, manage, and share files seamlessly. Built using Next.js 15, React 19, Appwrite, TypeScript, TailwindCSS, and ShadCN, it provides an efficient, secure, and user-friendly experience.

2. System Requirements

Ensure you meet the following requirements to use the platform:

- A modern web browser (Chrome, Firefox, Edge, Safari)
- A stable internet connection
- An active account on the platform

3. Getting Started

3.1 Creating an Account

1. Open the platform's URL.
2. Click on Sign Up.
3. Enter your details such as email and password.
4. Click Register to create an account.
5. Verify your email (if required) to activate your account.

3.2 Logging In

1. Navigate to the login page.
2. Enter your registered email and password.
3. Click Login to access your dashboard.

4. Using the Dashboard

The Dashboard is the main interface where you can view your storage usage, manage files, and access key features.

4.1 Understanding Storage Reports

- The dashboard displays a real-time storage usage report.
- A progress circle indicates the percentage of used storage.
- Storage is categorized into Documents, Images, Media, and Others.
- Recent uploads are listed with timestamps for easy access.

4.2 File Management

Uploading Files

1. Navigate to the Dashboard.
2. Click on the Upload button.
3. Select the file you wish to upload.
4. Confirm the upload; the file will be stored securely in Appwrite Storage.

Viewing and Managing Files

1. Navigate to the File Management Section.
2. Click on a file to preview or open it in a new tab.
3. Rename or delete files as needed.

Downloading Files

1. Locate the file in the file list.
2. Click the Download option.
3. The file will be saved to your device.

Deleting Files

1. Select the file you want to remove.

2. Click on the Delete option.
3. Confirm the action to permanently delete the file.

5. File Sharing

5.1 Sharing a File

1. Locate the file you want to share.
2. Click the Share button.
3. Enter the recipient's email or generate a shareable link.
4. Set access permissions (View, Edit, or Download).
5. Click Share to send the file.

5.2 Managing Shared Files

1. View shared files under the Shared Files section.
2. Modify permissions or revoke access when needed.

6. Search and Sorting Options

- Use the Global Search bar to find specific files.
- Sort files by name, date, or size to organize them efficiently.

7. Logout Settings

1. Access Logout button on the nav bar.
2. Click to logout when not in need.

9. Conclusion

The Storage and File Sharing Platform offers a modern, secure, and user-friendly way to manage files efficiently. Whether for personal use or collaboration, this system simplifies file organization and sharing with seamless cloud integration.

Start using the platform today and experience a hassle-free file management system!

CHAPTER NO. 07

CONCLUSION

7.1 Conclusion

The **Storage and File Sharing Platform** is designed to provide users with a seamless, secure, and efficient way to store, manage, and share files. Leveraging cutting-edge technologies such as React 19, Next.js 15, Appwrite, TypeScript, TailwindCSS, and ShadCN, the system ensures a modern and scalable approach to file management. Through features like real-time storage monitoring, role-based access control, file organization, and a user-friendly dashboard, the platform enhances collaboration and productivity. Rigorous testing methodologies, including unit testing, integration testing, and user acceptance testing, have been applied to ensure system stability, security, and performance.

7.2 Significance of the System

This platform plays a crucial role in **secure file storage and sharing**, offering multiple benefits to users:

- Enhanced Security – Secure authentication and access control prevent unauthorized access.
- Efficient File Management – Users can upload, organize, search, and download files effortlessly.
- Real-Time Storage Monitoring – Users can track available storage with detailed insights.
- Cross-Platform Compatibility – The responsive design ensures accessibility across devices.
- Improved Collaboration – Sharing features allow seamless collaboration among users.

By integrating Appwrite's backend services, the platform ensures data integrity and smooth file operations, making it a reliable solution for personal and professional use.

7.3 Limitations of the System

Despite its many advantages, the platform has certain limitations:

- Storage Capacity Restriction – The current implementation supports a 2GB storage limit per user, which may require future expansion.
- No Offline Access – The system relies on an active internet connection, limiting accessibility in offline scenarios.
- Limited File Preview Options – Currently, only basic file types (e.g., images, documents) can be previewed, while more advanced file formats (e.g., videos, PDFs) may require external applications.
- Scalability Challenges – As the number of users increases, database and storage optimization will be necessary to maintain performance.

7.4 Future Scope of the Project

To further enhance the system, several improvements and expansions can be made:

- Increased Storage Capacity – Implementing dynamic storage plans where users can upgrade their storage limits.
- Offline Mode – Introducing local storage caching to allow users to access files without an internet connection.
- Advanced File Preview – Adding in-platform viewers for PDFs, videos, and other complex file formats.
- AI-Powered File Search – Enhancing the search feature with AI-based indexing for faster and more accurate file retrieval.
- Mobile Application – Developing a mobile app version to improve accessibility and usability on smartphones and tablets.
- Blockchain Integration – Implementing decentralized storage and blockchain security for enhanced data privacy and tamper-proof records.

With continuous improvements and new technological integrations, the **Storage and File Sharing Platform** has the potential to become a robust, enterprise-level solution for secure and efficient file management.

References

For Learning and Help:

https://www.youtube.com/

https://nextjs.org/docs

https://appwrite.io/docs

https://www.typescriptlang.org/docs/handbook/

https://v2.tailwindcss.com/docs

Used Tools:

https://chatgpt.com/

https://www.perplexity.ai/

UI Inspiration:

https://www.uidesigndaily.com/

https://colorhunt.co/

For Ideas:

https://www.dropbox.com/

https://digiboxx.com/

https://www.box.com/en-in/

Plagiarism Report