File Sharing App

A PROJECT REPORT

Submitted by

Pankaj Tayal (23BCS13706)

Ayush Tiwari (23BCS11366)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING

Chandigarh University

Nov, 2025

BONAFIDE CERTIFICATE

Certified that this project report "**File Sharing App"** is the bonafide work of **"Pankaj Tayal"** and **"Ayush Tiwari"** who carried out the project work under my/our supervision.

SIGNATURE	**SIGNATURE**
Dr. Sandeep Singh Kang	
BATCH HEAD	**SUPERVISOR**
BE-CSE	**BE-CSE**

TABLE OF CONTENTS

INTRODUCTION

1.1. Introduction to Project

The Cloud Share application is a modern, full-stack web application designed for secure file upload, storage, and sharing. It demonstrates a robust, enterprise-level architecture by separating the application into distinct components. This project uses a sophisticated technology combination: **React.js** for the frontend, **Spring Boot (Java)** for the backend, and **MongoDB** for data persistence. To handle critical functions securely, it integrates specialized third-party services: **Clerk** for user authentication and **Razorpay** for payment processing and subscription management. The overall goal is to create a blueprint for a highly scalable and secure Software as a Service (SaaS) platform.

1.2. Identification of Problem

In today's digital environment, developers must move beyond simple database applications to build platforms that handle sensitive operations like file transfers and financial transactions. The challenge identified by this project is the need to combine stability and security (traditional enterprise concerns) with a modern, agile user experience (modern web development concerns). Specifically, a file sharing application requires.

1. A highly secure system for managing user identity and access control.
2. A transactional backend that can reliably manage file integrity and payments.
3. A scalable architecture that can handle a high volume of file metadata and user interactions.

LITERATURE REVIEW/BACKGROUND STUDY

2.1. Existing Solutions

Many existing file sharing solutions are either simple client-side applications (lack of security/scalability) or rely on rigid, older monolithic architectures. Enterprise-level systems must adhere to strict requirements for data integrity and security.

Backend Choice: The use of Java/Spring Boot is common in large-scale systems across industries like finance and healthcare. This preference is driven by Java's static typing, which improves reliability, and

Spring Boot's extensive support for complex transactional logic and robust error handling. This contrasts with lightweight, fast-iteration environments like Node.js, which are often less favored when long-term maintainability and deep stability are the primary focus.

Authentication Abstraction: Building custom authentication systems is time-consuming and prone to vulnerabilities. Modern best practice is to delegate this high-risk area to established services. The integration of Clerk reflects this trend, allowing the application to inherit enterprise-grade features like Multi-Factor Authentication (MFA) and biometric support.

2.2. Problem Definition

The core problem addressed by the Cloud Share architecture is **secure, decoupled file management for a commercial SaaS model.**

The solution requires overcoming three technical hurdles:

1. **Security Handshake:** Ensuring that the external identity provided by Clerk (who the user is) is flawlessly validated and integrated into the internal Spring Security context (what the user can access).
2. **File Vulnerability Mitigation:** File uploads are a high-risk vector for attacks (like remote code execution). The backend must implement strict protocols to sanitize and isolate uploaded content.
3. **Transactional Integrity:** The payment process (Razorpay) involves sensitive, multi-step asynchronous communication. The backend must reliably manage these transactions and prevent fraudulent subscriptions.

2.3. Goals/Objectives

The project objectives were defined to create a fully functional, production-ready application model:

Functional Objectives:

- ✓ **Core File Management:** Implement full Create, Read, Update, and Delete (CRUD) functionality for files via secure REST APIs.
- ✓ **Access Control:** Develop a feature to toggle files between public and private access statuses.
- ✓ **Secure Sharing:** Implement a mechanism to generate secure, shareable links for public files.
- ✓ **User Experience (UX):** Create a modern, responsive, and visually appealing user interface using React and Tailwind CSS, featuring both grid and list views for file display.

Non-Functional (Architectural and Security) Objectives:

- ✓ Managed Identity: Successfully integrate Clerk to manage all aspects of user registration, login, and session management, leveraging its security features.
- ✓ Secure Authorization: Configure Spring Security to validate the user’s identity (via Clerk-issued tokens) before allowing access to any file resource.
- ✓ Payment Integration: Successfully integrate Razorpay for subscription management, including secure Order Creation, Payment Verification via webhooks, and Transaction Persistence.
- ✓ Scalable Data Model: Utilize MongoDB's flexible schema to efficiently store and retrieve diverse file metadata, ensuring the system can scale horizontally to handle a large inventory of files and users.

DESIGN FLOW/PROCESS

❖ System Architecture

The system uses a highly decoupled **three-tier architecture**:

1. **Client (Presentation Layer):**

- ✓ **Technologies:** React.js, Tailwind CSS.
- ✓ **Role:** Handles all user interface rendering, user interaction, and state management. It communicates with the Server tier exclusively through REST API calls.

2. **Server (Business Logic and Persistence Layer):**

- ✓ **Technologies:** Spring Boot, Spring Security, MongoDB.
- ✓ **Role:** This is the core engine. It contains the business logic for file CRUD, authorization checks, and payment processing. Spring Security acts as the gateway, validating all incoming requests. MongoDB stores file metadata and transaction records.

3. **Third-Party Service Abstraction:**

- ✓ **Technologies:** Clerk (Authentication), Razorpay (Payments).
- ✓ **Role:** Offloads complex and high-risk operations to specialized, secure vendors. This simplifies development and ensures the application benefits from professional-grade security and compliance for identity and finance.

❖ Features

The following core features were implemented and tested:

Feature Category	Description
File Management	Uploading, viewing, downloading, and deleting files stored on the server.
Access Control	**Public/Private Toggle:** Ability for the file owner to instantly change the visibility status of any file.
Sharing	Secure generation of unique, obfuscated links for public files, allowing external users to access content without needing to log in.
User Identity	Full user management integrated via Clerk, handling sign-up, sign-in, and persistent user sessions.
Subscription	Integration of Razorpay to manage subscription tiers, enabling access to premium file storage or features upon verified payment.
Interface	Use of React and Tailwind CSS to create a modern and responsive dashboard supporting both grid and list views for file directories.

IMPLEMENTATION

❖ Implementation of solution

Backend Robustness (Spring Boot Rationale)

The backend relies on Spring Boot due to its suitability for enterprise-scale operations. It provides robust tools for managing complex tasks, especially financial transactions, where static typing in Java prevents many runtime errors common in dynamically typed languages. The framework's convention-over-configuration structure promotes maintainable, stable, and predictable applications, which is essential for a system handling data integrity.

Data Persistence (MongoDB)

MongoDB was chosen over traditional relational databases for file metadata storage. Its NoSQL, document-based structure allows for flexible and varied storage of data associated with each file (e.g., owner ID, access flags, share keys, version history). This flexibility simplifies schema evolution, which is common in rapidly developing SaaS products, and is optimized for the high-throughput read operations required to display large file dashboards efficiently.

Security and Authentication Deep Dive

1. **Clerk/Spring Security Integration:** Clerk manages the user's login session and issues a JSON Web Token (JWT) after successful authentication. The Spring Boot backend is configured to intercept every API request and validate this JWT. Once validated, Spring Security extracts the user's verified ID from the token. This ID is then used as the basis for all authorization decisions (e.g., "Does *this* user ID own *this* file ID?").
2. **File Upload Security Protocols (Mandatory Mitigations):** File sharing is a critical security nexus. The backend must enforce the following to mitigate risks like remote code execution:

- ✓ **Server-Side Validation:** Do not trust the file extension provided by the client. The backend must check the file's actual content type (MIME type) and validate it against an explicit whitelist of acceptable formats (e.g., allowing PDF but blocking PHP or JavaScript executables).
- ✓ **File Sanitization:** All uploaded files must be renamed using a universally unique identifier (UUID) generated by the server. This prevents path traversal attacks where an attacker attempts to trick the server into writing a file to a protected system directory.
- ✓ **Storage Isolation:** Files must be stored outside the server's publicly accessible web root directory. Files should only be accessed through a secured API endpoint that first checks authentication and authorization, not via direct URL access to the storage location.

Payment and Subscription Logic (Razorpay)

The payment module implements a high-stakes, three-step transactional workflow:

1. **Order Creation:** The Spring Boot API initiates a request to the Razorpay gateway to generate a unique order identifier for the user's subscription purchase.
2. **Payment Verification:** After the user completes payment on the Razorpay portal, Razorpay sends a secure, asynchronous message (webhook/callback) back to a dedicated Spring Boot API endpoint. This API is responsible for checking the transaction's integrity (e.g., verifying amounts and hashes) before *finally* activating the user's premium subscription status.
3. **Transaction Persistence:** Detailed logs of all order creation and verification events are securely stored in MongoDB to ensure a complete audit trail for billing and customer service purposes.

RESULTS ANALYSIS AND VALIDATION

The Cloud Share project successfully delivers a model application that goes beyond basic CRUD operations, validating its design choices and architectural maturity.

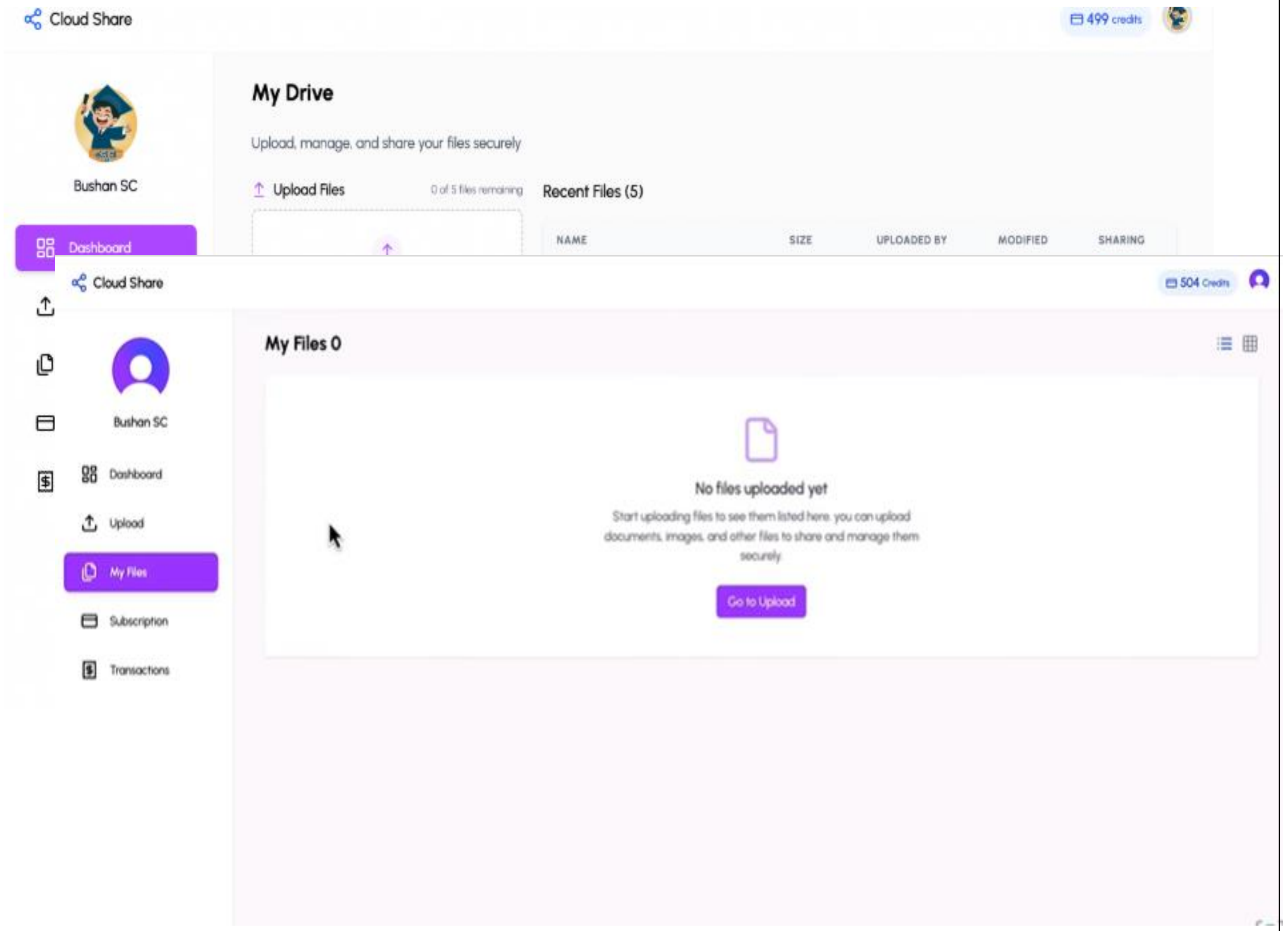
Architectural Validation:

- ✓ **Scalability:** The decoupled architecture (React/Spring Boot/MongoDB) confirms that the application is ready to scale. The frontend and backend can be deployed and scaled independently, and the use of MongoDB supports scaling the metadata layer horizontally.
- ✓ **Security Maturity:** By abstracting authentication to a service leader like Clerk, the development team demonstrates a mature understanding of risk management, ensuring the core security perimeter is battle-tested.
- ✓ **Portfolio Value:** For a developer, successfully integrating these high-level external services (Clerk for identity, Razorpay for payments) signals proficiency in complex, real-world development challenges related to security, scalability, and revenue generation—skills immediately valuable in the corporate and SaaS industries.

Feature Validation:

The successful implementation of core features, including the public/private file toggle and secure link generation, demonstrates mastery of both state management (frontend) and robust authorization logic (backend). The integration of payment functionality proves the system can handle complex, asynchronous transactional workflows, which is a key requirement for any commercially viable application.

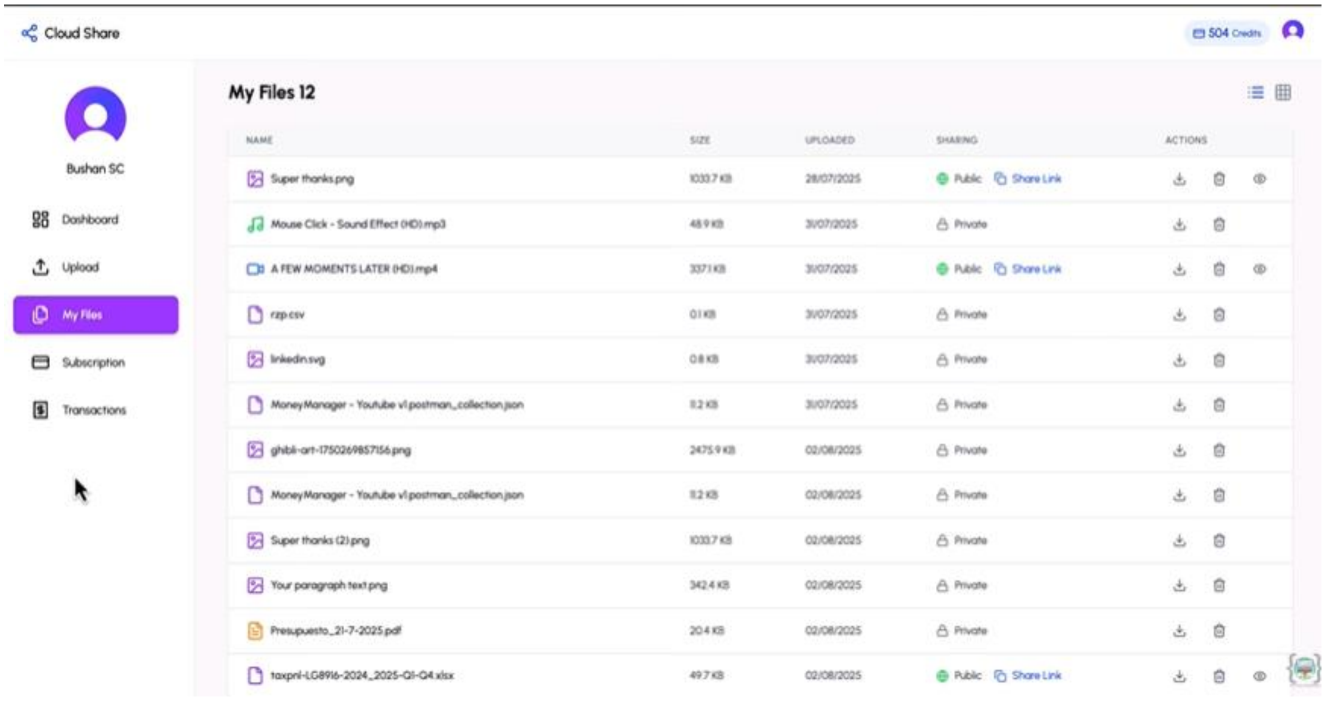
RESULTS/ OUTPUT:



Upload, manage, and share your files securely

No files uploaded yet

Start uploading files to see them listed here. you can upload documents, images, and other files to share and manage them securely



My Files 12

NAME	SIZE	UPLOADED	SHARING	ACTIONS
Super thanks.png	1033.7 KB	28/07/2025	Public Share Link	
Mouse Click - Sound Effect (HD).mp3	48.9 KB	31/07/2025	Private	
A FEW MOMENTS LATER (HD).mp4	337.1 KB	31/07/2025	Public Share Link	
rzp.csv	0.1 KB	31/07/2025	Private	
linkedin.svg	0.8 KB	31/07/2025	Private	
MoneyManager - Youtube v1.postman_collection.json	11.2 KB	31/07/2025	Private	
ghibli-art-1750269857156.png	2475.9 KB	02/08/2025	Private	
MoneyManager - Youtube v1.postman_collection.json	11.2 KB	02/08/2025	Private	
Super thanks (2).png	1033.7 KB	02/08/2025	Private	
Your paragraph text.png	342.4 KB	02/08/2025	Private	
Presupuesto_21-7-2025.pdf	20.4 KB	02/08/2025	Private	
taxpnl-LG8916-2024_2025-Q1-Q4.xlsx	49.7 KB	02/08/2025	Public Share Link	

CONCLUSION AND FUTURE WORK

6.1. Conclusion

The Cloud Share application serves as a strong architectural blueprint for a modern, production-grade file sharing service. The combination of the stable, enterprise-focused Spring Boot backend with the agile, modern React frontend, supported by managed security and payment services, provides a scalable and secure foundation. By prioritizing security protocols—especially around file upload and authorization—the project addresses the critical risks inherent in file management systems. The result is a highly authoritative and valuable portfolio piece demonstrating full-stack expertise.

6.2. Future work

To transition the application to a true high-volume production environment, the following architectural refinements are recommended:

1. **Externalize File Storage:** The application should be refactored to use a dedicated Object Storage service (such as AWS S3 or Azure Blob Storage) instead of relying on the local server's file system. This is mandatory for achieving true high availability and horizontal scalability of the file resources.
2. **Advanced Configuration Management:** All sensitive information (API keys, connection strings, and credentials) must be externalized from the application's configuration files. This should involve using secure environment injection mechanisms suitable for cloud deployment, such as Kubernetes Secrets or dedicated configuration services.
3. **Investigate Reactive Architectures (Spring WebFlux):** Since file uploads and downloads are high-concurrency tasks, exploring a transition of these core APIs to a non-blocking, reactive model (like Spring WebFlux) would significantly enhance the application's performance and throughput under heavy concurrent load, mitigating potential bottlenecks in the traditional Spring Boot architecture.
4. **Strengthen Input Validation:** Ensure strict input validation is implemented across all data fields received from the frontend to guard against common vulnerabilities like Cross-Site Scripting (XSS).