BASED ON MINIO CLOUD DRIVE APPLICATION DESIGN AND DEVELIOPMENT

By

YI QIAO

A Project Report

Submitted to the Department of Computing & Software And the School of Graduate Studies

of McMaster University

in Partial Fulfilment of the Requirements for the degree

Master of Engineering

McMaster University

© Copyright by Prachi Goyal, December 2024

Master of Engineering (2024) (Computing & Software)

McMaster University Hamilton, Ontario, Canada

**TITLE:** BASED ON MINIO CLOUD DRIVE APPICATION DESIGN AND DEVELOPMENT

**AUTHOR:** Yi Qiao

M.Eng. Student (Computer & Software) McMaster University, Hamilton, Canada

**SUPERVISOR:** Dr. Richard Paige

**NUMBER OF PAGES:** 38

# ABSTRACT

With the acceleration of digital transformation, cloud storage services have become an integral part of personal and enterprise data management. This paper researches and implements a web storage system based on MinIO, Spring Boot, and Vue.js, aiming to provide a secure, efficient, and easy-to-manage data storage solution. The system utilizes the high-performance object storage capability of MinIO, combines the rapid development features of Spring Boot and the interactive front-end framework of Vue.js. The entire system supports basic file operations such as uploading, downloading, sharing, and previewing, while SSL encryption is used during data transmission to protect the security of user data. A responsive and user-friendly netdisk application was created.

This thesis details the requirement analysis, system design, function implementation and evaluation. In future work, this research will explore advanced technologies including second transfer, encrypted transmission optimization, and WebSocket protocol to further enhance system performance and user experience.

Keywords: cloud storage, MinIO, Spring Boot, Vue.js, web storage system, data security, user experience

CONTENTS

[ABSTRACT 3](#_Toc83111243)

[1. INTRODUCTION 5](#_Toc986290551)

[1.1 Background and Significance of the Study 5](#_Toc159019464)

[1.2 The Need for Research 5](#_Toc1170474580)

[1.3 Research Content and Objectives 5](#_Toc1216059540)

[1.4 Research Methodology and Technical Route 6](#_Toc710820281)

[1.5 Organization of the paper 6](#_Toc304934506)

[2. RELATED WORK 8](#_Toc1138260600)

[2.1 Overview of Cloud Storage Technology 8](#_Toc961576724)

[2.2 Existing Cloud Storage Services 8](#_Toc1405556593)

[2.3 Open-Source Cloud Storage Solutions 9](#_Toc869541551)

[2.4 MinIO: High Performance, Kubernetes-Native Object Storage 9](#_Toc758629822)

[2.5 Vue.js: An Incremental JavaScript Framework 10](#_Toc681006115)

[3. REQUIREMENTS 12](#_Toc1729419942)

[3.1 Requirements Gathering 12](#_Toc169803049)

[3.2 Functional Requirements 12](#_Toc2021561327)

[3.3 Non-functional Requirements 13](#_Toc1042443702)

[4. DESIGN 15](#_Toc1179707288)

[4.1 Overall Design 15](#_Toc1771360312)

[4.2 Domain Detail Design 17](#_Toc686965423)

[4.2.1 User and Auth Domain 17](#_Toc955778089)

[4.2.2 File Domain 18](#_Toc584662263)

[4.2.3 Preview Domain 21](#_Toc1680969216)

[5. IMPLEMENTATION 23](#_Toc1902237027)

[5.1 DDD Development 23](#_Toc1308659900)

[5.1.1 Code Structure 23](#_Toc119426726)

5[.1.2 Overall ER Diagram 24](#_Toc1455257584)

[5.2 Functional Implement 25](#_Toc822958605)

[5.2.1 User Domain 25](#_Toc1670587555)

[5.2.2 File Domain 27](#_Toc1363836007)

[5.2.3 Preview Domain 29](#_Toc1898805218)

[5.2 Interface Development 30](#_Toc1612304506)

[6. EVALUATION 32](#_Toc1053174496)

[6.1 Writing Test Cases 32](#_Toc1143535698)

[6.2 Writing and Structuring Testing 32](#_Toc1573319283)

[6.3 Unit Testing 33](#_Toc811043870)

[6.4 End-to-End Testing 33](#_Toc1135615581)

[6.5 Requirements Coverage 33](#_Toc1603898978)

[7. CONCLUSION 34](#_Toc1515386102)

[7.1 Future Work 34](#_Toc2085646541)

[A. REFERENCES 35](#_Toc85844606)

# INTRODUCTION

## 1.1 Background and Significance of the Study

In the contemporary era of rapid development of information technology, data, and information have become the blood of social operation. How to store and access these data efficiently and securely has become an important driving force for technological progress. Cloud storage service, as one of the solutions, is rapidly gaining popularity among individual and enterprise users with its unique advantages. In particular, open-source cloud storage solutions, such as Minio, provide users with customized service options to adapt to changing storage needs and challenges. Focusing on the application of open-source cloud storage services, this study explores the design and implementation of a cloud disk application based on Minio, aiming to provide users with a data storage option with high performance, high reliability, and easy management.

## 1.2 The Need for Research

Although there are many cloud disk services on the market today, they are often one-size-fits-all solutions that lack sufficient flexibility to meet the individual needs of specific user groups. For example, enterprise users may need to deploy cloud services in their internal network environment to ensure data privacy and security; research institutions may need customized data analysis tools combined with storage solutions; and individual users may seek more efficient data synchronization and backup functions. Existing cloud drive offerings often fail to provide adequate customization support in these areas.

In addition, many cloud disk services have limitations in terms of data sovereignty, with users' control over their own data restricted by the service provider's policies and technical architecture. Due to the ever-changing laws and regulations and the increasing demand for data sovereignty from enterprises and individuals, autonomous and controllable cloud disk services have become an inevitable trend. Based on these real-world needs, the development of a Minio-based autonomous cloud disk application not only provides customized services but also improves the flexibility and efficiency of data processing while ensuring data sovereignty and security.

## 1.3 Research Content and Objectives

The main goal of this thesis is to develop a web disk application that integrates the functions of disconnected transfer, file encryption, user management, and so on. The research covers the whole process from requirement analysis, system design, and interface implementation to functional testing. The back-end development of the system will be in Java and use the Spring Boot framework to improve development efficiency and simplify the deployment process. The front-end interface will be realized by the Vue.js framework to ensure the responsiveness and interactivity of the user interface. This research will also delve into the implementation of the breakpoint transfer technology and how to effectively manage files and user data in an online disk application to provide a secure data transfer and storage solution.

## 1.4 Research Methodology and Technical Route

Object Storage Service (OSS) is a massive, secure, low-cost, and highly reliable cloud storage service suitable for storing any type of files. Capacity and processing capacity are elastically expandable, and multiple storage types are available for selection, fully optimizing storage costs.AliCloud Object Storage OSS (Object Storage Service) is a massive, secure, low-cost, highly persistent cloud storage service provided by AliCloud. Its data is designed to be no less than 99.999999999999% (12 9s) persistent, and service availability (or business continuity) is no less than 99.995%.

MinIO is an object storage service based on the Apache License v2.0 open-source agreement. It is compatible with Amazon S3 cloud storage service interface, ideal for storing large-capacity unstructured data, such as images, videos, log files, backup data and containers/virtual machine images, etc., and an object file can be any size, from a few kilobytes to a maximum of 5T ranging.MinIO is a very lightweight service that can be easily integrated with other applications, such as NodeJS, Redis, or MySQL. For small and medium-sized enterprises, Minio is a good choice if you don't want to go to the cloud for storage. Minio can be used directly as object storage, but also as a gateway layer for object storage services on the cloud, seamlessly connecting to Amazon S3, and MicroSoft Azure.

In order to realize the research objectives, this paper adopts the method of combining theoretical research and empirical analysis. Firstly, a literature review is conducted to analyze the current state of development of cloud storage technology and netbook applications and determine the entry point of the research. Subsequently, the system functions are determined through requirement analysis, and the system architecture is designed based on the characteristics of Minio. In the implementation phase, this research will follow the agile development principle to iteratively complete the development and integration of each functional module. System testing will cover unit testing, integration testing, and performance testing to ensure the stability and reliability of the application.

## 1.5 Organization of the paper

This paper is organized as follows: chapter 1 introduces the background of the research, the need for the research, the content and objectives, and the research methodology. Chapter 2 overviews the related technologies and theoretical foundations, including cloud storage technologies, features of Minio, and the technology stack used for development. Chapter 3 analyzes the requirement points in detail. Chapter 4 discusses system design in detail, including architecture design, functional planning, interface definition, and data model. Chapter 5 shows the system implementation process, including development environment setup, code writing, functional implementation, and interface design. Chapter 6 conducts system testing, analyzes the test results, and evaluates the system performance. The last chapter summarizes the whole paper and presents an outlook on the future research direction.

# RELATED WORK

## 2.1 Overview of Cloud Storage Technology

Cloud storage technology has evolved dramatically since its inception, driven by the growing demand for data accessibility and disaster recovery options. Initially, the concept of cloud storage was to provide users with remote servers where they could store their data without having to worry about maintenance and physical hardware issues. Over time, these services have evolved to provide not only storage but also processing power, allowing complex applications and services to be fully hosted in the cloud. This evolution can be traced back to the development of virtualization technologies, which abstract the physical hardware to allow multiple virtual machines to run on a single physical server [[1]](#endnote-0)[1]. The scalability of these systems is made possible by distributed architectures, which allow data to be stored in multiple locations, thereby increasing redundancy and reliability. As the amount of data generated by organizations continues to grow exponentially, these cloud storage technologies have become an integral part of data management strategies. The shift from capital expenditure (CAPEX) to operational expenditure (OPEX) models has also been a significant factor in the adoption of cloud storage solutions, allowing businesses to pay only for the storage they use, rather than investing in expensive hardware infrastructures [[2]](#endnote-1)[2].

The fundamental properties of cloud storage, such as on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service, have been detailed by leading researchers in the field and form the basis for modern cloud computing paradigms[[3]](#endnote-2)[3]. These features contribute to the widespread adoption of cloud storage technologies as they offer flexibility and scalability that cannot be matched by traditional storage solutions. In addition, the ability to access data anytime anywhere has revolutionized the way businesses operate, enabling new methods of workflow and collaboration. The importance of these technological advances cannot be overemphasized, as they pave the way for the next generation of Internet services and applications.

## 2.2 Existing Cloud Storage Services

The landscape of cloud storage services is dominated by several key players, each offering their unique take on cloud storage. Amazon Web Services (AWS) introduced the concept of cloud storage to the masses with its Simple Storage Service (S3), which remains a benchmark for durability, availability, and scalability in the industry [[4]](#endnote-3)[4].AWS S3 provides an object storage service with an impressive 99.99999999999% durability as well as comprehensive security and compliance features, making it the storage solution of choice for organizations that need a robust storage solution for organizations that need a robust storage solution. Microsoft’s Azure Blob Storage complements its cloud offerings by providing a service that integrates seamlessly with other Azure services, offering options for hot, cool, and archive data storage, catering to various business needs[[5]](#endnote-4)[5]. Google Cloud Storage has made great strides by tightly integrating with its data processing services, especially in the area of data analytics, thus appealing to organizations looking to leverage big data[[6]](#endnote-5)[6].

Despite the power of these services, they are not without challenges. Vendor lock-in is a major issue, as the unique features and APIs offered by each vendor can make it difficult for customers to migrate data to different services. Additionally, the cost implications of data transfer and manipulation can be complex and sometimes unpredictable.Complicating the situation is the evolving nature of data protection laws, which require cloud storage providers to continually adapt their services to meet regulatory requirements. As a result, the balance between innovation, cost and compliance remains an evolving goal for the cloud storage industry.

## 2.3 Open-Source Cloud Storage Solutions

The proliferation of open-source cloud storage solutions has reinvigorated the storage market, providing alternatives that prioritize transparency, customizability, and community-driven development. These solutions cater to a diverse set of needs, from personal cloud storage to enterprise-level deployments. Ceph, for instance, is a unified, distributed storage system designed for excellent performance, reliability, and scalability. It is often used in situations that require highly scalable block, file, and object storage under a single whole-system namespace[[7]](#endnote-6)[7]. On the other hand, extends Kubernetes functionalities, turning distributed storage systems into self-managing, self-scaling, and self-healing storage services, thereby simplifying the deployment and management of storage solutions in cloud-native environments [[8]](#endnote-7)[8].

The open-source model also encourages innovation in cloud storage technology. It enables organizations to deploy and customize their own storage solutions to fit specific use cases, which is not always possible with proprietary services. The community development approach enables rapid iteration and incorporates cutting-edge features such as erasure coding and geo-replication, which improves data durability and availability. In addition, the ability to inspect and modify source code provides an additional layer of security and trust, as any security vulnerabilities are quickly identified and addressed by the community.

## 2.4 MinIO: High Performance, Kubernetes-Native Object Storage

MinIO has become an important player in the cloud-native object storage space, offering high performance and compatibility with the S3 API, which many organizations have found critical to meeting their cloud storage needs. Designed from the ground up to support private cloud and containerized environments, MinIO is well-suited for a wide range of data-intensive applications, from machine learning to big data analytics. Its design philosophy centers on simplicity and performance, with a single-layer architecture that facilitates straightforward scaling and management[[9]](#endnote-8)[9] .

MinIO is an object storage service based on the Apache License v2.0 open-source protocol that can be used for cloud storage solutions to save massive amounts of images, videos, and documents. The server side can work on Windows, Linux, OS X, and FreeBSD due to Golang implementation. Configuration is simple, basically copying the executable program, single line commands can be run up.MinIO is compatible with the Amazon S3 cloud storage service interface, which is ideal for storing large-capacity unstructured data, such as images, videos, log files, backup data, and container/virtual machine images, etc., and an object file can be of any size, ranging from a few kilobytes to a maximum of 5T. Its suitability for high-throughput, low-latency applications has been demonstrated in a variety of industry and academic environments, demonstrating its ability to handle the workloads required by modern applications while maintaining ease of use and deployability[[10]](#endnote-9)[9].

At the core of MinIO's support for distributed deployments and high availability of services and data is MinIO's Codec Correction feature. MinIO implements Codesmithing as a core component to provide data redundancy and availability. Assuming that MinIO divides an object into K data slices, and deletion correction generates M checksum slices based on the K data slices, MinIO needs at least K slices of any type to recover the original object. MinIO requires at least K slices of any type to recover the original object, meaning that M slices can be allowed to fail out of a total of K+M slices[[11]](#endnote-10)[10].

MinIO's approach to security is also worth mentioning. The service provides robust security features, including end-to-end encryption, identity and access management, and the ability to create fine-grained access control policies. These features ensure that MinIO can be deployed in sensitive environments where data security is critical. In addition, MinIO's open-source nature provides a vibrant community of developers and users who contribute to the ongoing development of MinIO and provide support through community forums and documentation[[12]](#endnote-11)[11].

## 2.5 Vue.js: An Incremental JavaScript Framework

Vue.js has emerged as an incremental JavaScript framework for building user interfaces. Unlike other monolithic frameworks, Vue was designed from the ground up to be incrementally adoptable. Its core library focuses only on the view layer, making it easy to integrate with other libraries or existing projects. Vue is also perfectly capable of powering sophisticated Single-Page Applications (SPAs) when used in combination with modern tooling and supporting libraries[[13]](#endnote-12)[12].

The flexibility of Vue.js allows developers to build applications to their liking, which has earned the framework a large following in the developer community. Vue.js has a gentle learning curve compared to more complex frameworks, making it popular with both novice and experienced developers, and its extensive documentation and active community support have further contributed to its widespread adoption. Inspired by Google developer Evan You, Vue.js was created in 2014 and inspired by Angular. Like Angular, it is a JavaScript-based toolkit system that is used to build a dynamic user interface. It is progressive, scalable, and best of all, open-source, so there are lots of third-party instruments to play with.[[14]](#endnote-13)[13].

# REQUIREMENTS

## 3.1 Requirements Gathering

Requirements gathering is an exploratory process that involves researching and documenting the exact needs of the project from start to finish. Effective requirements gathering and requirements management starts at the beginning of the project. Requirements gathering is one of the most essential parts of any project and can add value to the project on multiple levels. Gathering, understanding and managing requirements is a key factor in the success of a software development effort[[15]](#endnote-14)[18].

Competitor Research and Analysis: In the first phase of requirements gathering, the focus is on competing products in the market, such as Google Cloud Drive, Amazon Cloud Drive, and Baidu Cloud Drive. through in-depth research on the functional features, user interface design, market positioning, and user feedback of these competing products, we can gain industry best practices and insights. insights into the industry's best practices. For example, Google Cloud Drive is known for its powerful collaboration features and extensive application integration; Amazon Cloud Drive offers stable performance and excellent enterprise services; and Baidu Cloud Drive may have a larger user base in the domestic market. After collecting this information, we can analyze the demand points and pain points of our target user groups and develop a differentiated strategy to provide a clear positioning direction for our Cloud Drive product.

## 3.2 Functional Requirements

User Registration and Login Functionality Design: The first interaction of a user is usually the registration process, so we need to design a registration process that is simple and intuitive, yet secure enough. Users should be able to register with an email address or cell phone number and verify their identity with a verification code. After registration, the user will be logged in, and the login process needs to have certain security checking mechanisms, such as restrictions and prompts when the password is entered incorrectly. For the convenience of users, we can also provide a social media account login to simplify this process.

User authentication (forensics): Once a user is registered and logged in, the system needs to ensure that only authenticated users have access to the resources they have permission to access. This requires the implementation of a forensic system that verifies the user's identity when they perform sensitive operations such as changing passwords, accessing private files, etc. The implementation of an authentication system may use technologies such as OAuth 2.0 or JSON Web Tokens (JWT), which provide secure management of user sessions and ensure the security of interactions.

User personal information management: Users should be able to view and edit their personal information in the personal account center, such as changing passwords, updating personal information, and managing bound social media accounts. This part requires a good user interface design to ensure that users can easily perform various operations. Meanwhile, for changes to sensitive information, the system should verify the user's identity again to prevent unauthorized access.

User rights and roles management: In enterprise-level applications, different users may have different rights and roles. For example, administrator users need to have permission to manage general user accounts, assign user roles, and access various advanced settings of the system. General users, on the other hand, are limited to accessing personal and shared files. The management of roles and permissions needs to be flexible and extensible so that when new roles are added in the future or the permissions of existing roles are modified, the system can support them without major changes.

The core of Cloud Drive is to provide a stable and efficient file storage and transfer platform. File uploading and downloading are the most basic functions, but users may encounter unstable networks or other interruptions when transferring large files, so the function of intermittent transfer is particularly important, which can save the progress of the transferred files when the transfer is interrupted and continue the transfer automatically or manually when the network is restored, which can significantly improve the user experience. In the user module, in addition to the basic registration and login functions, it is also necessary to provide user rights management, password recovery, and user profile editing functions to meet the needs of different users. As for file management, not only basic file operations should be realized, but also how to manage files effectively, such as file version control, sharing settings, and cross-device synchronization should be considered.

## 3.3 Non-functional Requirements

System Scalability: As the number of users and file sizes continue to grow, the system must be able to smoothly scale its resources to handle larger storage and transmission requirements. This involves not only the scalability design of back-end storage, such as the application of distributed file systems but also how the front-end efficiently handles the display and management of large file lists. We need to ensure that these issues are taken into account in the design of the system architecture, using e.g. a microservices architecture to keep the system modular and flexible.

Split file uploading is a key technology that makes the process of uploading large files much more efficient and stable. By splitting large files into multiple smaller chunks of data for uploading, the system can make better use of network bandwidth while only having to re-upload the unsuccessful portion, rather than the entire file, in the event of network instability or outage. This approach not only optimizes the data transfer process but also provides additional benefits in terms of file security, as separate blocks of data can be encrypted separately, enhancing data security. When implementing this technique, careful consideration needs to be given to issues including slice size, data recovery mechanisms after transmission interruptions, and how to efficiently reorganize the slice on the server side.

When considering the file search function, efficiency and accuracy become core considerations. In order to achieve fast and accurate file searching, an efficient indexing system needs to be constructed that is capable of handling a large number of file attributes and content indexes. The design of the search system needs to ensure that results are returned quickly even in large datasets, which may involve complex search algorithms and user interface design so that users can perform file searches based on different criteria, such as file name, type, or modification date. The search function must be implemented with the user experience in mind while taking into account back-end performance and scalability to cope with growing data volumes.

Distributed Minio, the cornerstone of cloud storage, is designed to provide a high degree of data redundancy and system availability. By decentralizing data storage across multiple server nodes, Minio ensures that even if some nodes fail, the data remains secure and the service is sustainable. Such a system design requires efficient data synchronization mechanisms to ensure that data remains consistent across all nodes. At the same time, the system's load-balancing mechanism must be able to intelligently distribute requests to optimize the resource utilization and response speed of the entire network. In addition, the distributed storage solution needs to be scalable so that more storage nodes can be added seamlessly as the business grows.

# DESIGN

## 4.1 Overall Design

Implementing Domain-Driven Design presents a top-down approach to understanding domain-driven design (DDD) in a way that fluently connects strategic patterns to fundamental tactical programming tools.[[16]](#endnote-15)[14] In Domain-Driven Design (DDD), domains are delineated based on the boundaries of the business logic, aiming to confine the complexity of the software to a specific context, thus making development and maintenance more manageable.Therefore, in this thesis, I will use DDD for modeling.

One of the fundamentals of DDD is that we choose a model (by which we mean a system of abstractions, not a UML diagram or other concrete artifact) well suited to the problem at hand. Yet a legacy system already has an established model, albeit implicit, and this model can seldom be changed with a reasonable amount of effort. Even if the legacy model could be changed, the new model might not suit the legacy functionality -- the change could undermine what the legacy system was always good at[[17]](#endnote-16)[15].

In domain-driven design, the process of domain delineation begins with a deep understanding of business knowledge. Through close collaboration with business experts, a common language is created to ensure conceptual consistency. Next, conceptual models are constructed by identifying key events and operations in business processes that capture and organize key concepts and rules in the business domain. Implementing context mapping helps to clarify the boundaries between different models and define how they interact. This delineation reflects the natural boundaries of the business domain and provides internal consistency and external autonomy of the models through bounded contexts[[18]](#endnote-17)[16].

Domain delineation is important because it reduces complexity and improves model clarity. Models within bounded contexts encapsulate specific business logic and allow teams to develop and maintain their respective parts of the system independently, thereby increasing system flexibility and maintainability. In addition, this delineation allows different teams to use context-specific language and models within clearly defined boundaries, which reduces ambiguity and enhances communication efficiency across teams. Ultimately, this approach supports a fine-grained understanding of business requirements and enables easier adaptation and evolution as the business grows and the market changes.

Therefore, I divided the four main domain, which are:

* **User Center**: The User Center centralizes all user-related operations. According to the principles of DDD, user identity and state management are part of the core domain because they are often directly related to business rules. The User Center, as a separate bounded context, can focus on user lifecycle management, which is consistent with business strategy requirements and layered architecture principles.
* **File Domain**: Theoretical Support: File Management is a technology-driven support domain that involves the storage, retrieval, and processing of files, operations that require specific technologies and storage solutions. DDD recommends a modular architecture that allows the complexity of file management to be encapsulated within this domain, reducing direct interaction with other domains, such as user management, and allowing for independent expansion and optimization.
* **Authentication Domain**: Authentication is a cross-cutting concern of a system that ensures its security. Separating it out allows the authentication logic to be independent of the business logic, following security best practices. A separate authentication service means that a dedicated team can be responsible for security issues, reducing the risk of security breaches, and allowing more flexibility to interface with different authentication mechanisms and standards.
* **Preview Domain**: Preview functionality may involve the rendering and transformation of different file types, which often have different performance and scalability requirements than CRUD operations on files. The Preview Domain can focus on providing the user with an immediate view of the contents of a file without having to deal with other lifecycle events of the file, following the Principle of Focus and the Principle of Single Responsibility.

Based on the above description of domains and domain contexts according to DDD, and in conjunction with my business model, I have depicted the following domain design model:

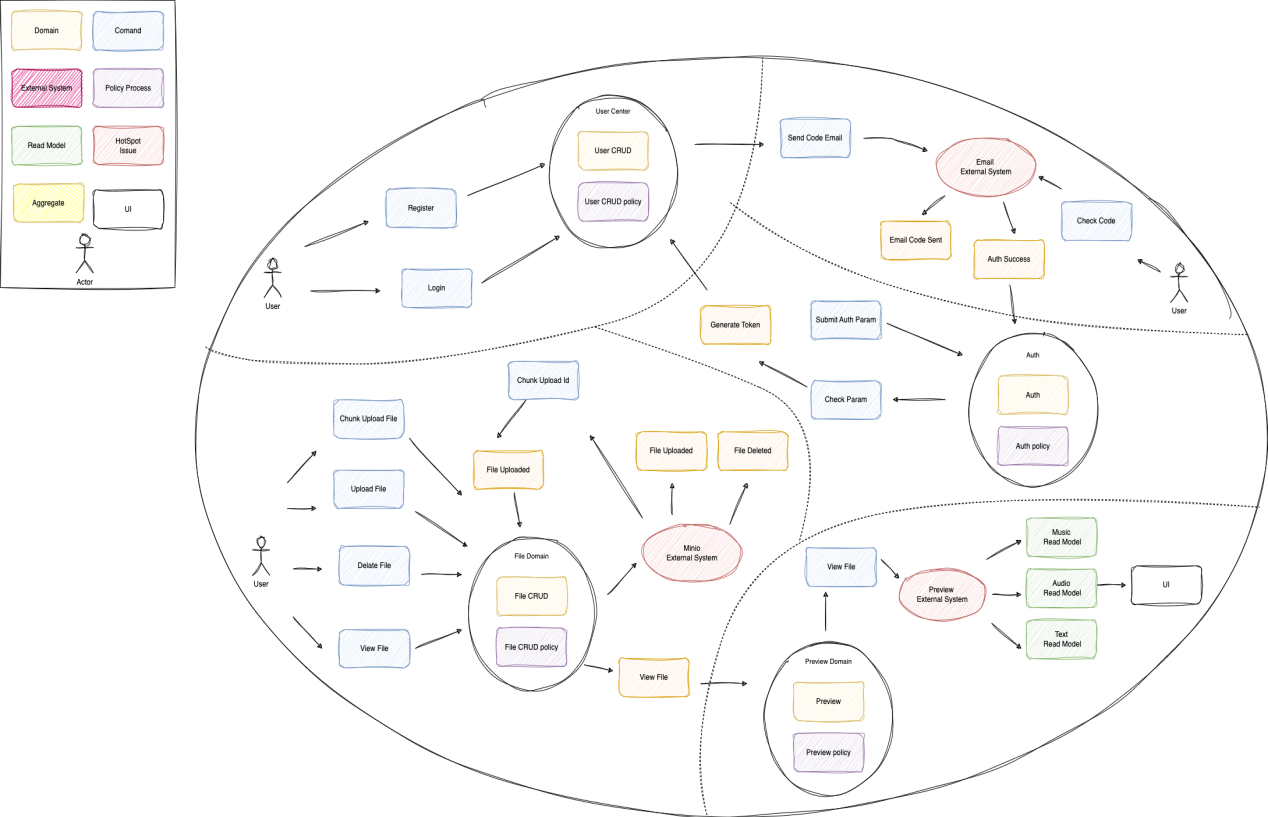


Figure 4.1 Overall DDD Entity and Context Design

## 4.2 Domain Detail Design

### 4.2.1 User and Auth Domain

#### 4.2.1.1 User Register

This sequence diagram describes the process of the user registration process and the interactions of the system. The process begins when the user visits the registration page and then enters a username, password, and e-mail address in a form and submits it. The registration information is sent to the authentication service, which checks the uniqueness of the username and e-mail. Once the information is confirmed to be unique, the authentication service requests the token service to generate an activation token.

Next, the mail service sends an email containing the activation token to the user for authentication. The user clicks on the activation link in the email and this link and token are verified by the authentication service. Once the activation link and token are verified, the user's account is activated and the authentication service notifies the user that the account was successfully activated and provides a login page.

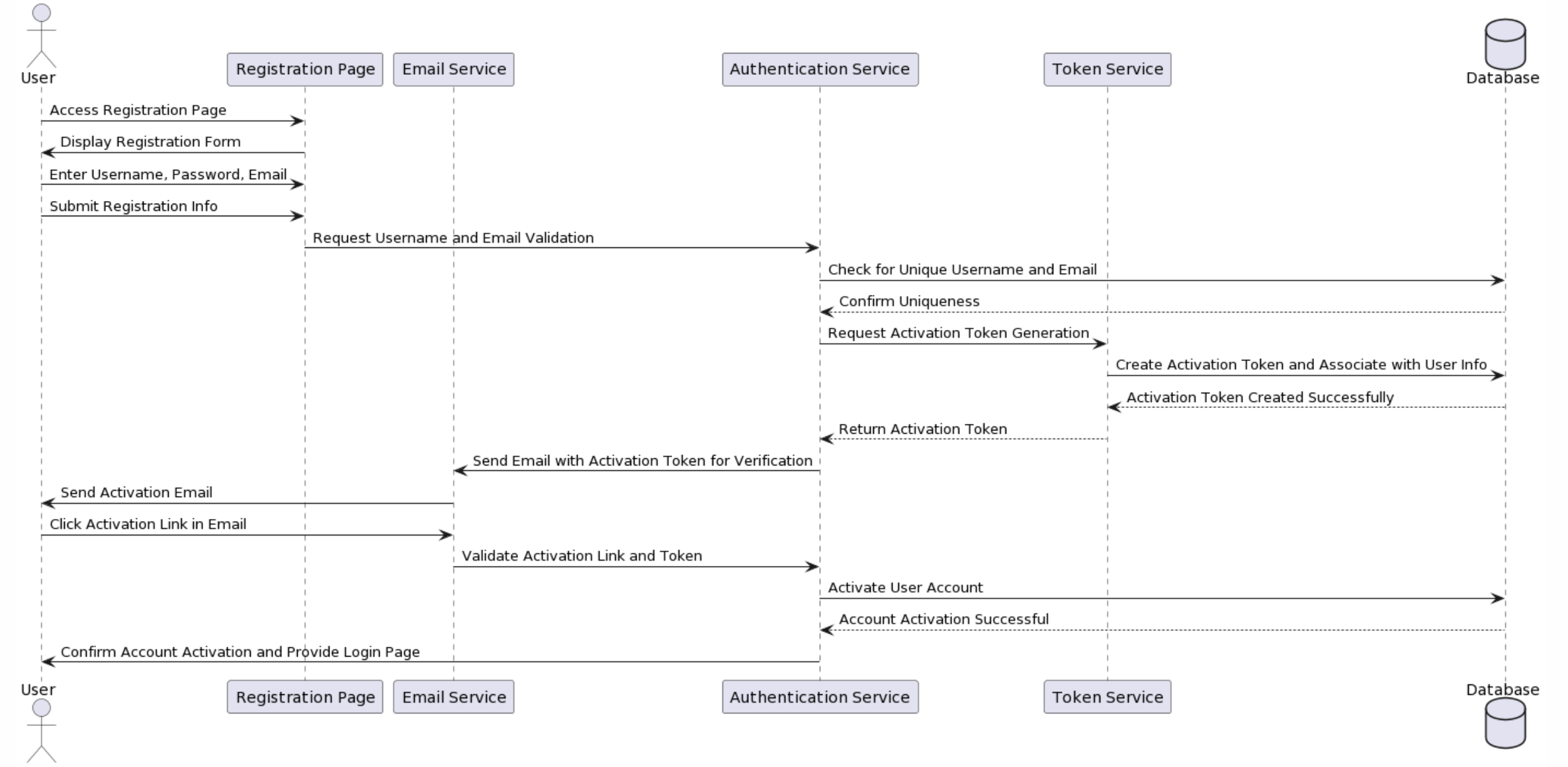


Figure 4.2 User Register Sequence Diagram

#### 4.2.1.2 User Login

This sequence diagram describes the process of the user logging into the MacDrive system. The user first visits the login page, which requests and displays a CAPTCHA.

By default, Kaptcha is very easy to setup and use and the default output produces a captcha that should be fairly hard to bust. The captcha's it produces by default look very similar to the one above. If you would like to change the look of the output, there is several configuration options and the framework is modular so you can write your own morphing code[[19]](#endnote-18)[17].

The user enters a username, password, and the CAPTCHA they see, and may select the "Remember Me" feature. Once the login information and the CAPTCHA are submitted, the authentication service verifies that the CAPTCHA is correct. Once the CAPTCHA is verified, the authentication service continues to verify the user name and password provided by the user. If the user information is verified correctly, the authentication service requests the token service to generate a token, which is then created and stored. Finally, after the user is authenticated and successfully logged in, the system displays the user's dashboard or user area. Throughout the process, the database plays the role of storing and verifying information, ensuring the security and accuracy of the login process.

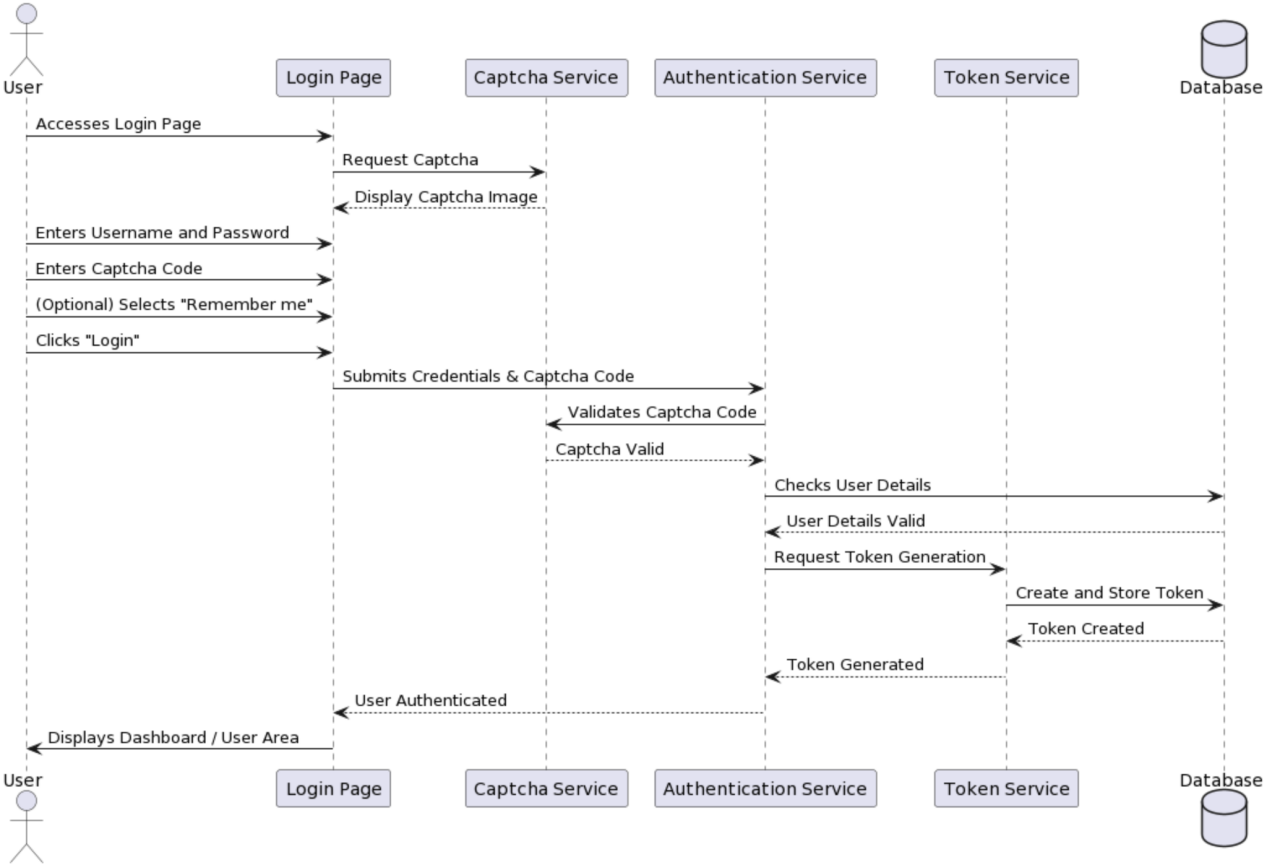


Figure 4.3 User Login Sequence Diagram

### 4.2.2 File Domain

#### 4.2.2.1 File Upload

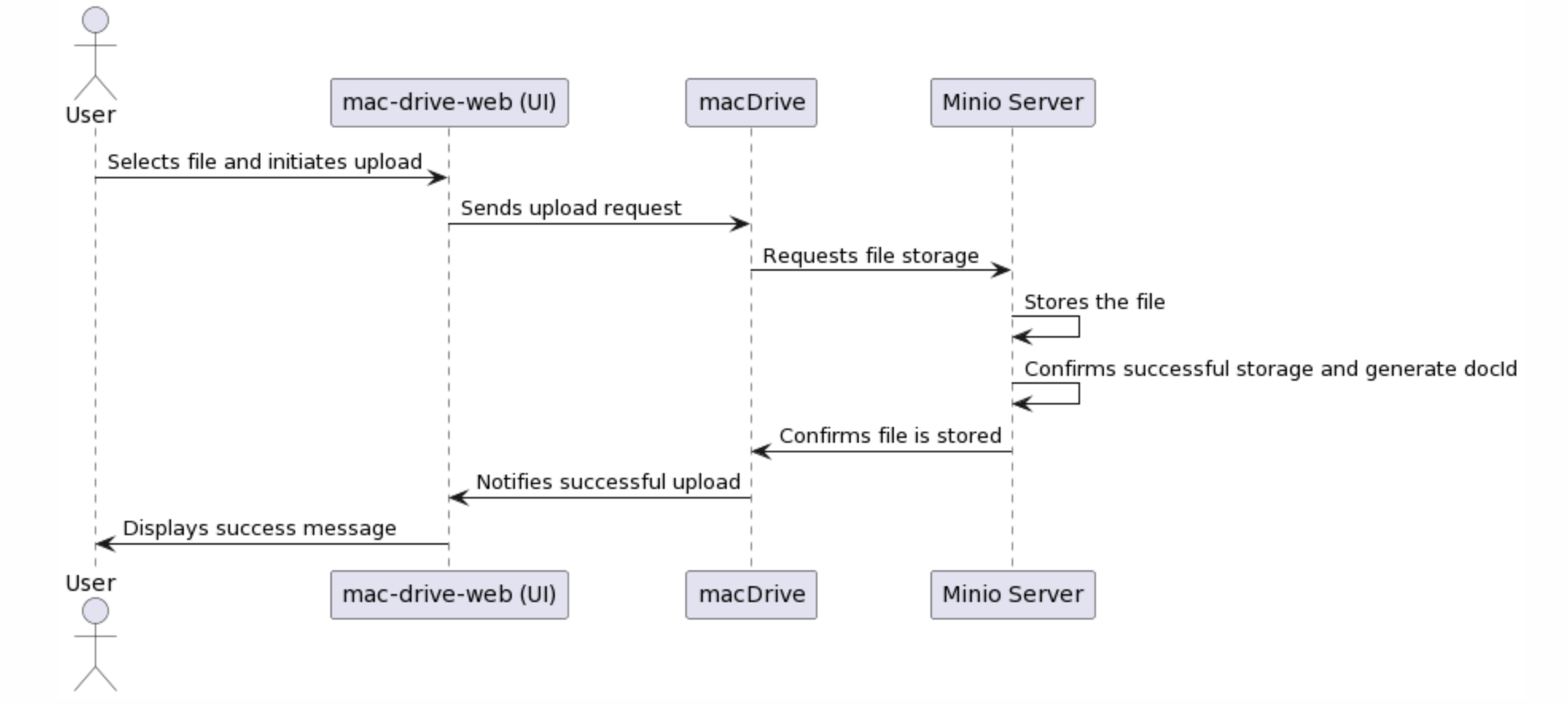
This sequence diagram shows a user uploading a file. User first selects the file and initiates the upload process. From front-end mac-drive-web receives the user's upload request and makes a request to the back-end macDrive to store the file. The macDrive then forwards the request to the Minio server, which is actually responsible for storing the file. After the file is successfully stored, the Minio server confirms to the macDrive and generates a unique document ID (docId). MacDrive then sends the confirmation of the successful file storage back, which then displays a success message to the user informing them that the file has been successfully uploaded. 

Figure 4.4 User Upload File Sequence Diagram

#### 4.2.2.2 File Chunk Upload

Chunked upload is a technique for uploading large files that optimizes the entire upload process by splitting the file into smaller chunks. This method can significantly improve the reliability and efficiency of large file uploads. When the network is unstable or bandwidth is limited, slice uploads allow individual failed segments to be re-uploaded without having to start the entire file upload from scratch. Parallel transmission of these slices also allows better utilization of network bandwidth and speeds up the upload process. In addition, this approach improves upload resilience because if an upload is interrupted, it can be continued from the last successfully uploaded fragment rather than re-uploading the entire file.

From a user experience perspective, the chunk uploads provide a more granular indication of progress, allowing users to see the progress of the upload for each fragment rather than the entire file. This instant feedback gives users a better sense of the upload process, especially when uploading large files. On the server side, processing small segments compared to large files allows for smoother allocation of resources and reduces bursts of stress on the server.

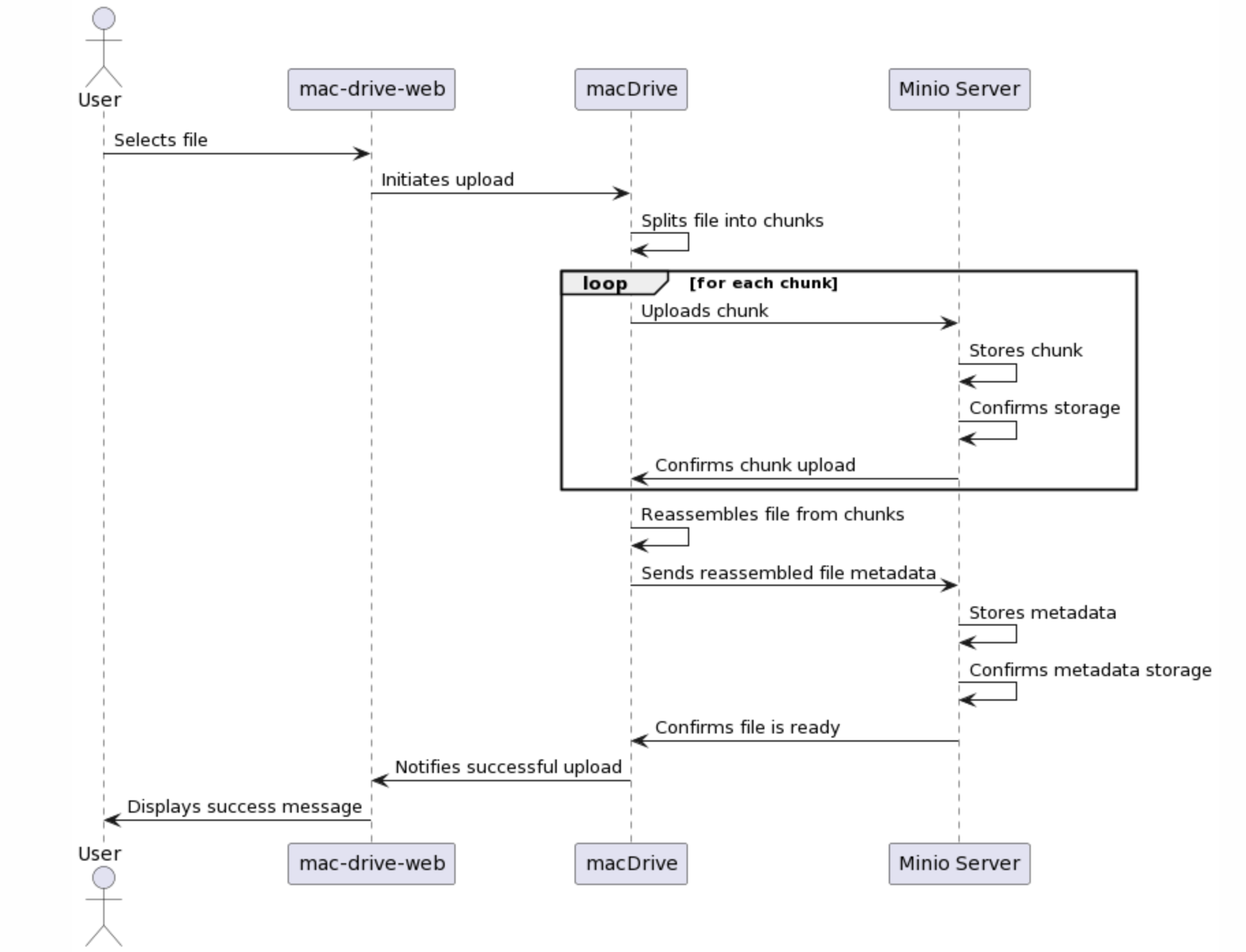
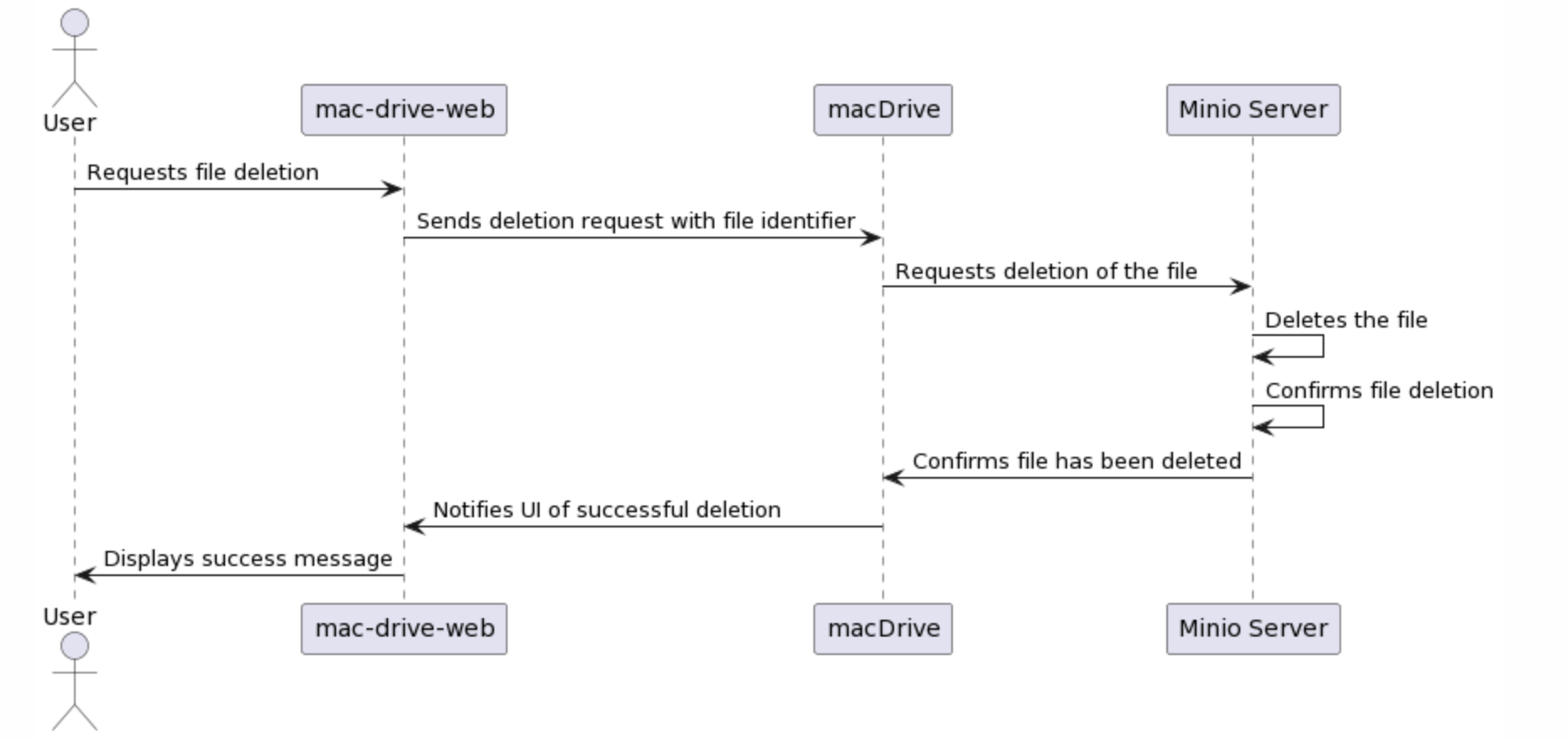
The sequence diagram describes the process of file uploading by the user through the mac-drive-web interface. After the user selects a file, the upload process begins. macDrive splits the file into multiple chunks. Then, macDrive enters a loop to upload each block. For each file chunk uploaded, the Minio server stores the chunk and confirms that the storage was successful. macDrive, after uploading all the chunks and receiving confirmation, reassembles the chunks into a complete file and sends the reassembled file metadata to the Minio server. the Minio server stores the metadata and confirms that the metadata was stored successfully. Once the entire file and its metadata have been confirmed for storage, macDrive notifies the mac-drive-web interface that the file is ready. Eventually, the mac-drive-web interface displays a success message to the user notifying them that the file upload was successful. 

Figure 4.5 User Chunk Upload File Sequence Diagram

#### 4.2.2.3 File Delete

The sequence diagram shows a user deleting a file through a web interface. The user initiates a request to delete a file from the mac-drive-web interface, carrying the file identifier. This request is passed to the macDrive, which then sends a delete command to the Minio server, which performs the deletion and confirms to the macDrive that the file has been deleted. macDrive receives the confirmation and notifies the web interface that the file was deleted successfully, and the web interface ultimately displays a success message to the user.

Figure 4.6 User Delete File Sequence Diagram

#### 4.2.2.4 File Display

This sequence diagram shows the user requests to view a list of files through the mac-drive-web interface. To process this request, mac-drive-web first initiates a request to macDrive to view the files. macDrive requests authentication from the user, and after the user provides an authentication token, macDrive uses this token to initiate an authentication request to the Minio server to obtain the list of files. the Minio server retrieves the list of files and returns it to macDrive. macDrive then sends the list of files back to the macDrive interface. back to macDrive. macDrive then sends the list of files back to the mac-drive-web interface, which eventually presents the list of files to the user.

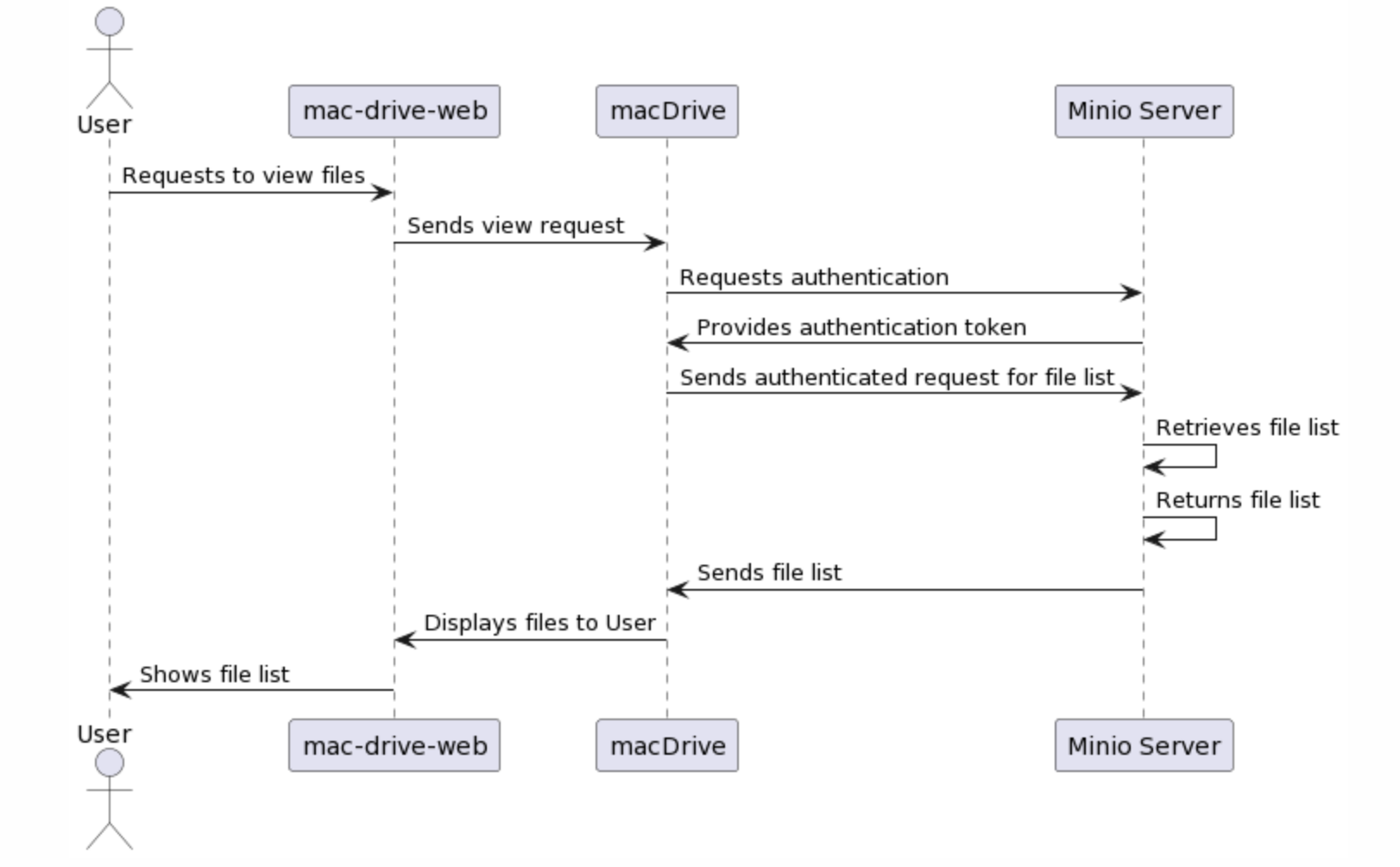


Figure 4.7 File Display Sequence Diagram

### 4.2.3 Preview Domain

This sequence diagram shows a user initiates a request from the mac-drive-web interface to view files stored on the system. mac-drive-web forwards this request to macDrive, which requests that the user authenticate. After the user provides authentication, macDrive sends an authenticated request to the Minio server to retrieve the list of files. The Minio server responds to this request, retrieves the list of files, and returns it to macDrive. macDrive then requests a third-party service to process the file types to recognize and process the file types such as music, video, and documents in the file list. After processing the file types, the third-party service returns the processed information to macDrive. The macDrive receives the processed file type information and then sends the file list with file type icons back to the mac-drive-web interface. Finally, the mac-drive-web interface displays the list of files with icons to the user, thus enabling the user to visually identify and browse files according to their types.

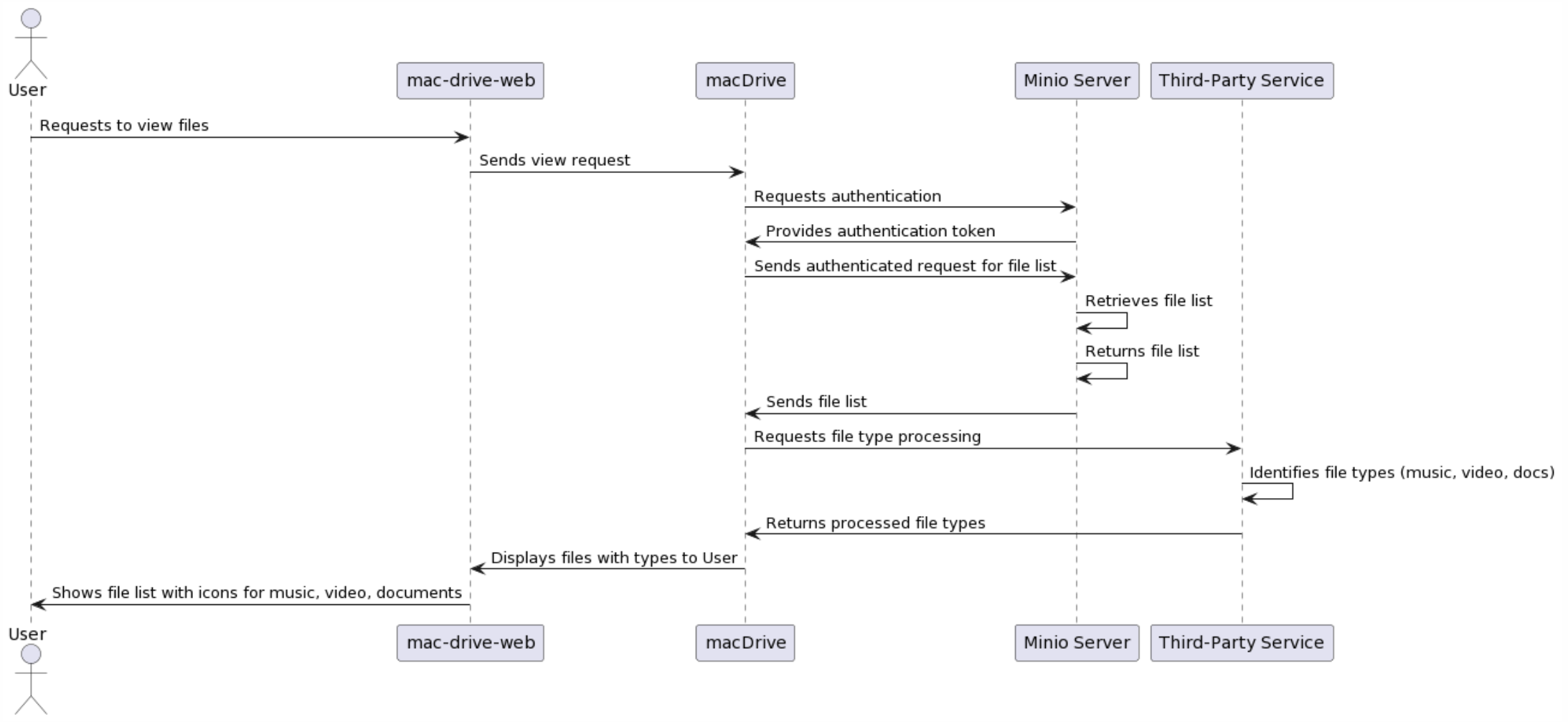


Figure 4.8 Preview Domain Sequence Diagram

# IMPLEMENTATION

## 5.1 DDD Development

### 5.1.1 Code Structure

Eric proposed the layered architecture in 2003. Compared with the traditional three-tier architecture of "Presentation Layer + Business Logic Layer + Data Access Layer", there is one more layer, and the main difference is that the Business Logic Layer is divided into Application Layer and Domain Layer[[20]](#endnote-19)[20].

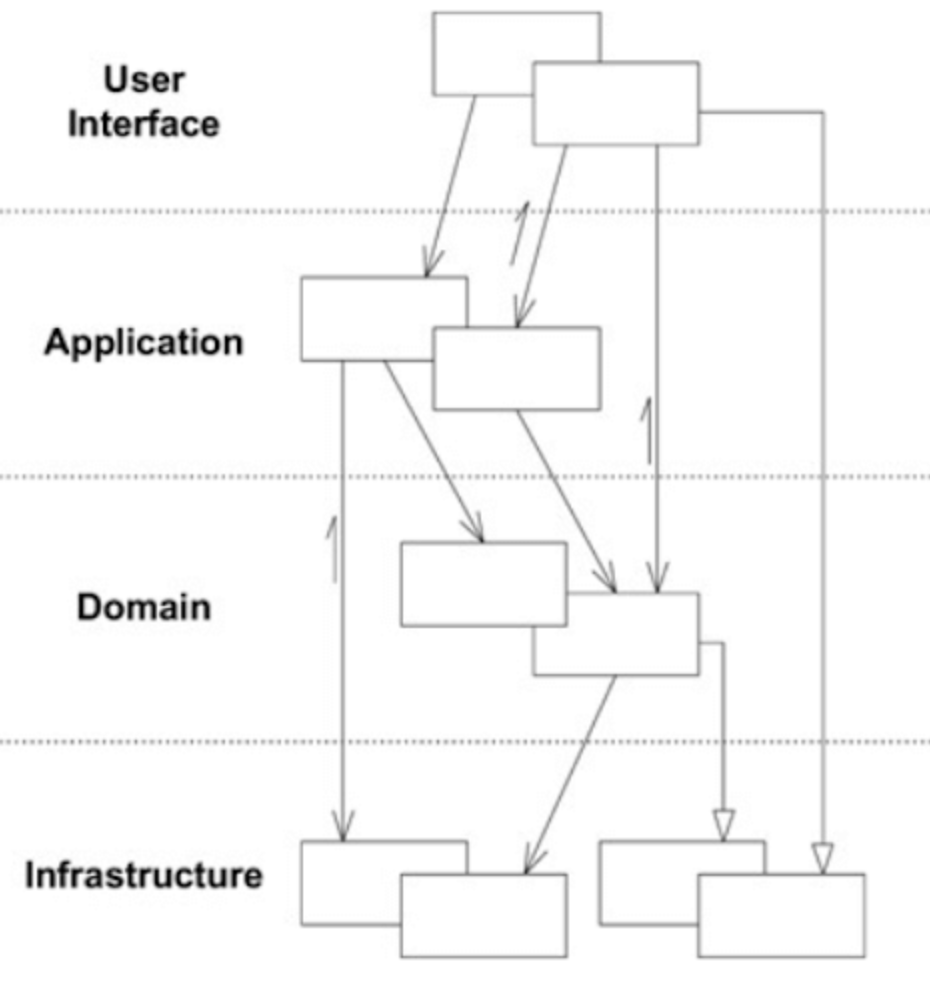


Figure 5.1 DDD Layer[[21]](#endnote-20)[20]

I will refer to this layering scheme proposed by Eric for the architectural design of the service. I Section 4.1 Overall Design, I divided the four main domain, which are:

**User Center, File Domain, Authentication Domain, Preview Domain；**



Figure5.2 macDrive project Layer

### 5.1.2 Overall ER Diagram

In this system, each user has a unique identifier (Id) and various information associated with his/her identity, such as Name, Email, Password, Avatar and Storage Capacity. Each user is associated with a specific Authority, which defines the scope of the user's operations in the system, such as uploading, downloading or sharing files, reflecting the role-based access control mechanism.

In terms of the File entity, it contains key attributes such as File Name, File Type, File Path, File Storage idWork Name, unique identifier (id), Upload Time and File Size. The key attributes are File Path, File Storage idWork Name, unique identifier (id), Upload Time, and File Size. Files are linked to File Class through the "Contain" relationship, which is further subdivided into different File Extension Name and File Type Id, such hierarchical categorization helps users to This hierarchical classification helps users to organize and retrieve files.

The relationship between the user and the file is linked by "Operate", which allows the user to perform multiple operations on the files he/she owns. The relationship between a file and its category is represented by the "Contain" edge, which indicates the relationship between a file and its category. The entire model is user-centric, revolving around the user's ability to perform operations and the organization of files, which constitutes the data skeleton of the Cloud Drive system.

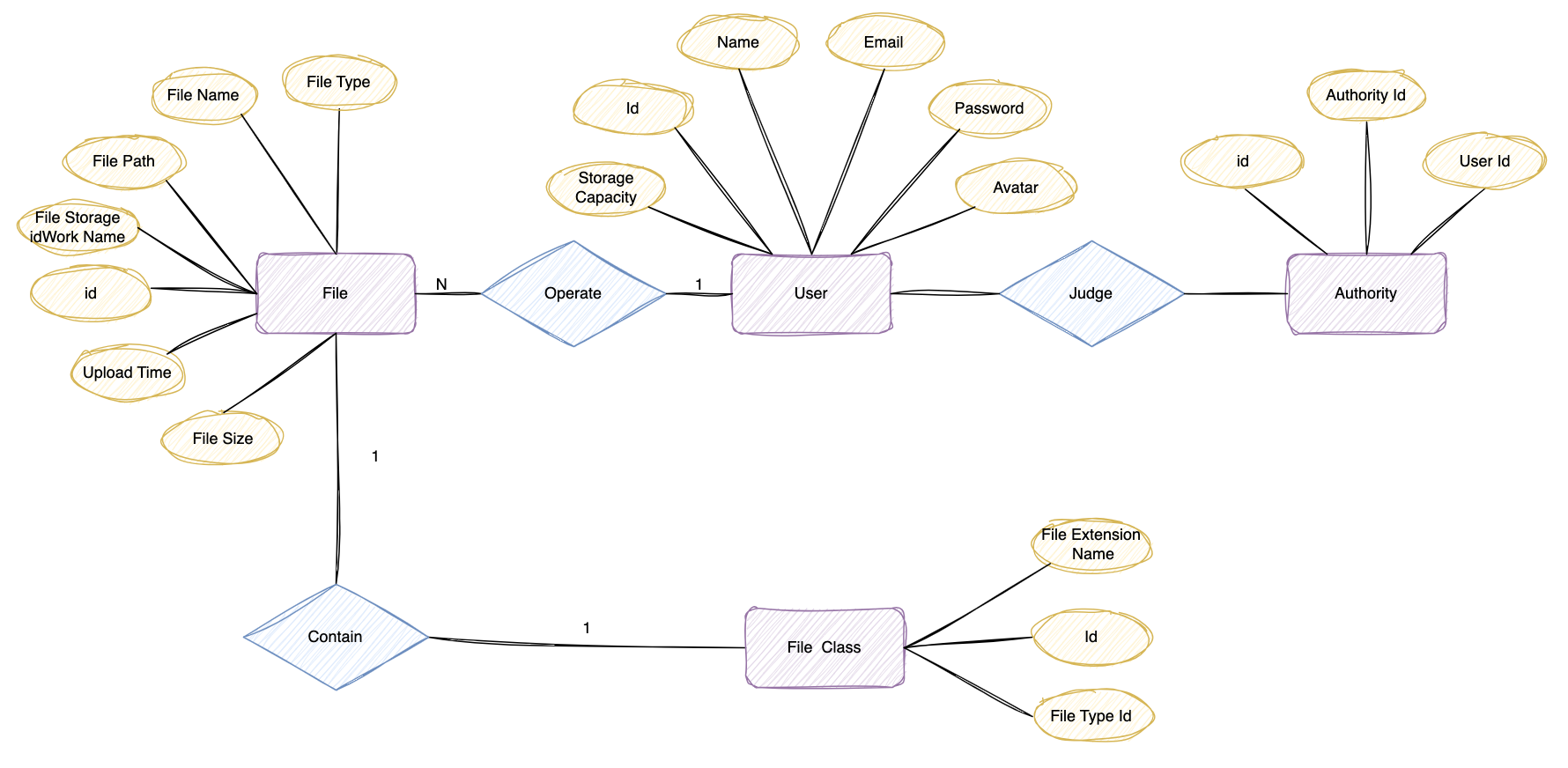


Figure 5.3 Overall ER Diagram

## 5.2 Functional Implement

## 5.2.1 User Domain

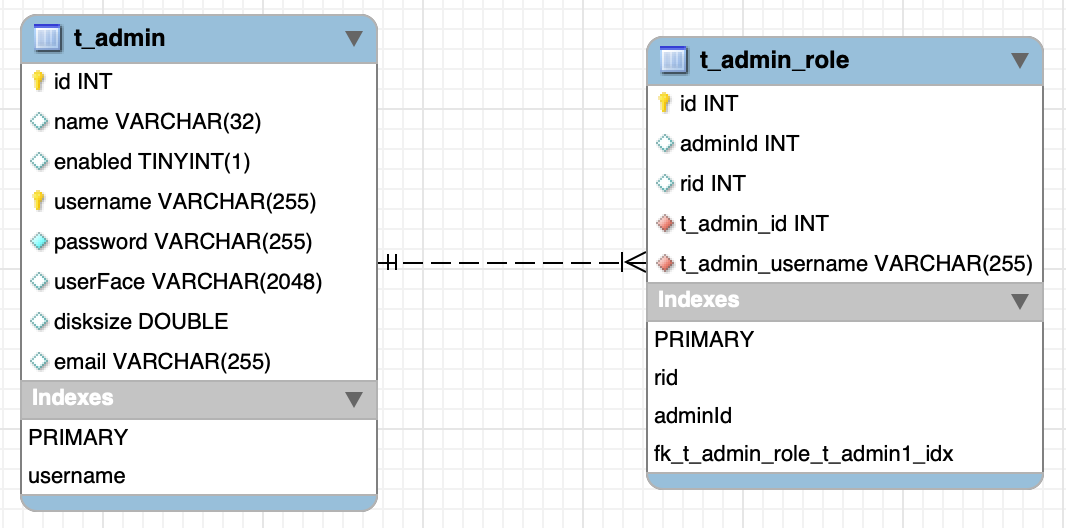


Figure 5.4 User ER Detail Diagram

The user domain I store through two tables, t\_admin and t\_admin\_role. Figure 5.4 shows the user account information and its relationship to the roles. The "t\_admin" table in the database design is dedicated to storing the user's personal information and account settings. It records the user's name, username, password and other information required for authentication, as well as some personalized information, such as avatar and disk space size. These fields not only provide the netdisk system with a comprehensive record of the user's personal information, but also allow necessary system configurations, such as assigning different storage limits to different users.

Meanwhile, the "t\_admin\_role" table shows how the system assigns various roles to different users. Each role is identified by a unique "rid" and associated with a user's "id", which allows the system to flexibly define the scope of responsibilities and permission levels for different users. This role assignment mechanism is the key to ensure that the system can accurately control and manage administrator privileges.

The two tables of users are connected by the "t\_admin\_id" and "t\_admin\_role\_adminId" fields, which indicates that the system considers the consistency and integrity of the data when handling the administrator role assignment. For example, when the administrator information is updated in the "t\_admin" table, these changes are automatically reflected in the relevant records in the "t\_admin\_role" table, ensuring the synchronization of role and administrator information. This detailed and precise database design is the basis for realizing an efficient and secure web hosting service.

The following screenshot shows the actual implementation of the system, which are 5.5 User Login, 5.6 User Register, 5.7 User Info Detail：

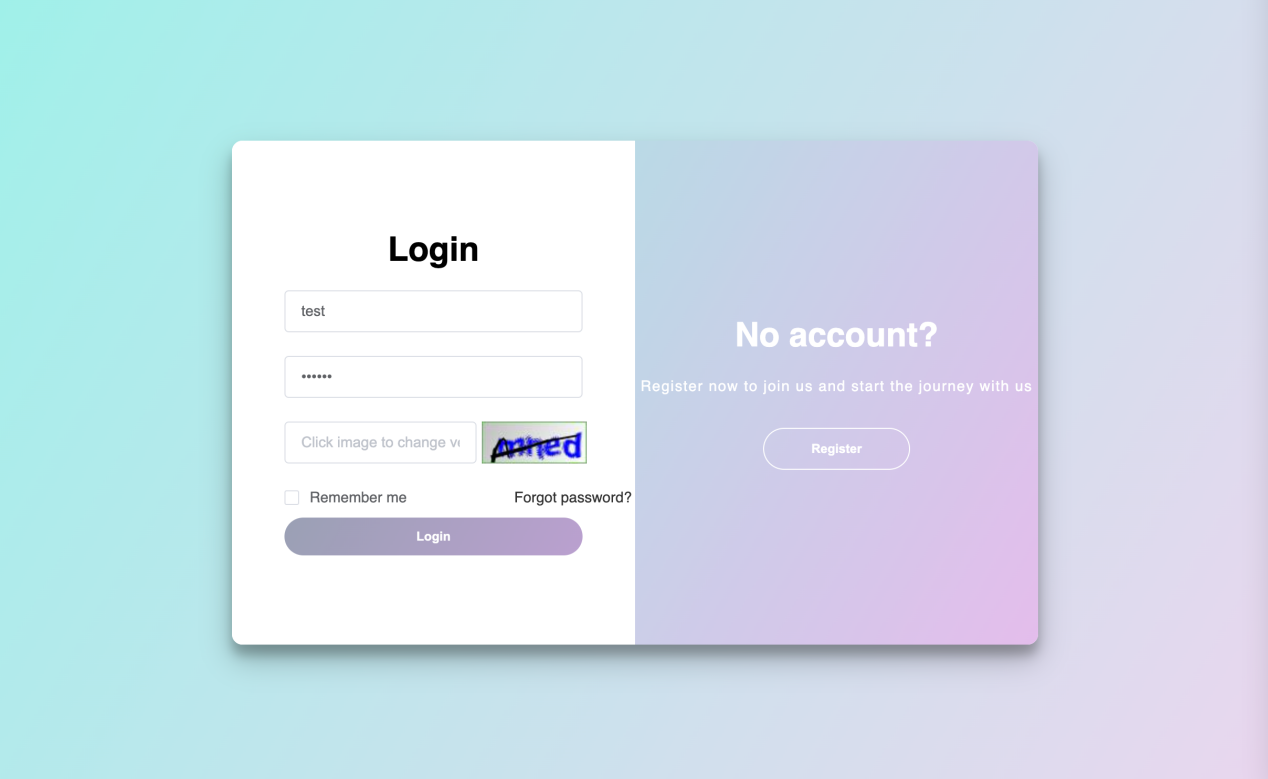


Figure 5.5 User Login Page

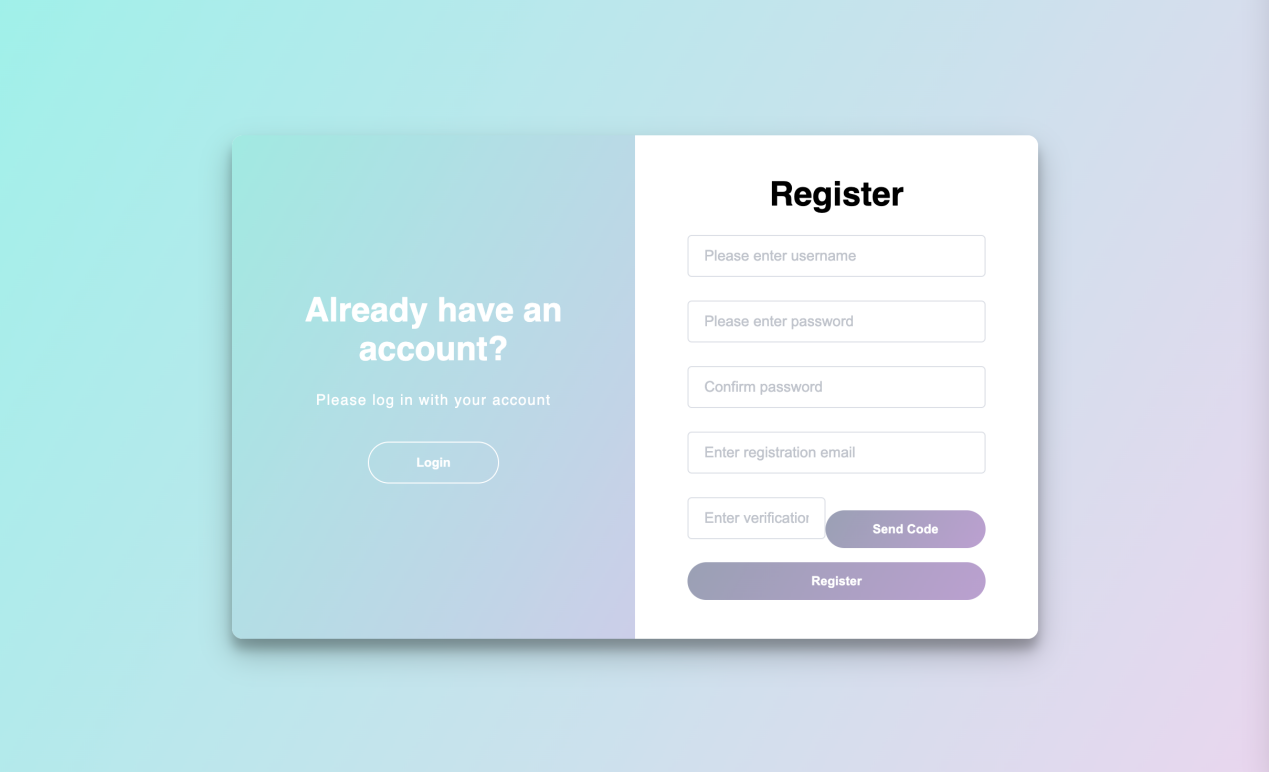


Figure 5.5 User Register Page

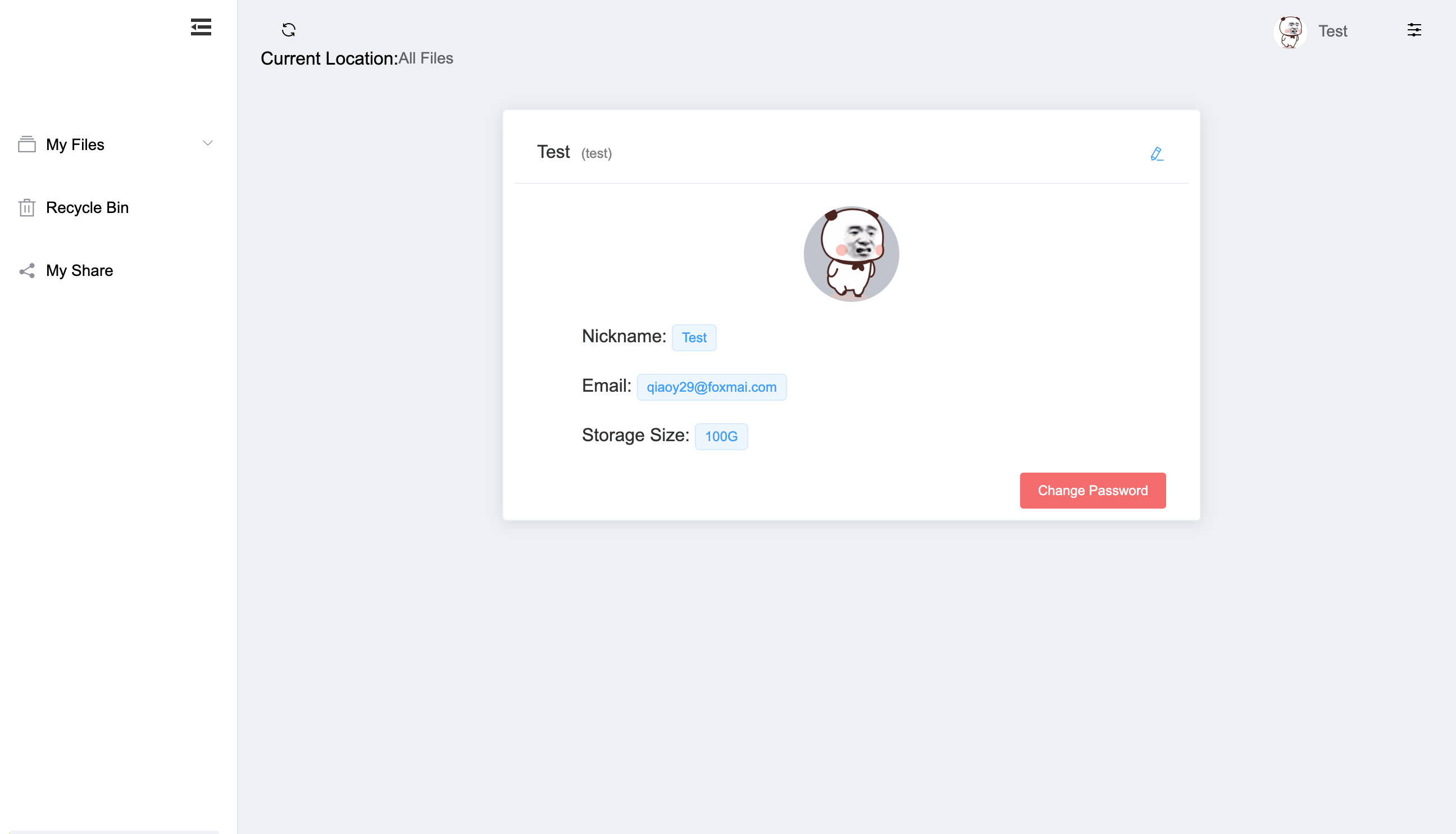


Figure 5.6 User Infomation Detail Page

## 5.2.2 File Domain

The file itself is defined in the "t\_file" table, which records the file's name, type, storage information, creation and deletion dates, and so on. These detailed records not only ensure file tracking and management, but also facilitate file operations such as deletion and restoration. The "t\_file" table is also associated with the "t\_fileclass" and "t\_filetype" tables through foreign keys, with the former recording the class to which the file belongs, and the latter further specifying the type of the file.

"The "t\_fileshare" and "t\_filechunk" tables handle file sharing and chunk uploads, respectively. The records in the "t\_fileshare" table provide the expiration time and the sharing link for each sharing operation, while the "t\_filechunk" table records the detailed information about the chunks of the file upload, such as the number of chunks and the size of each chunk.

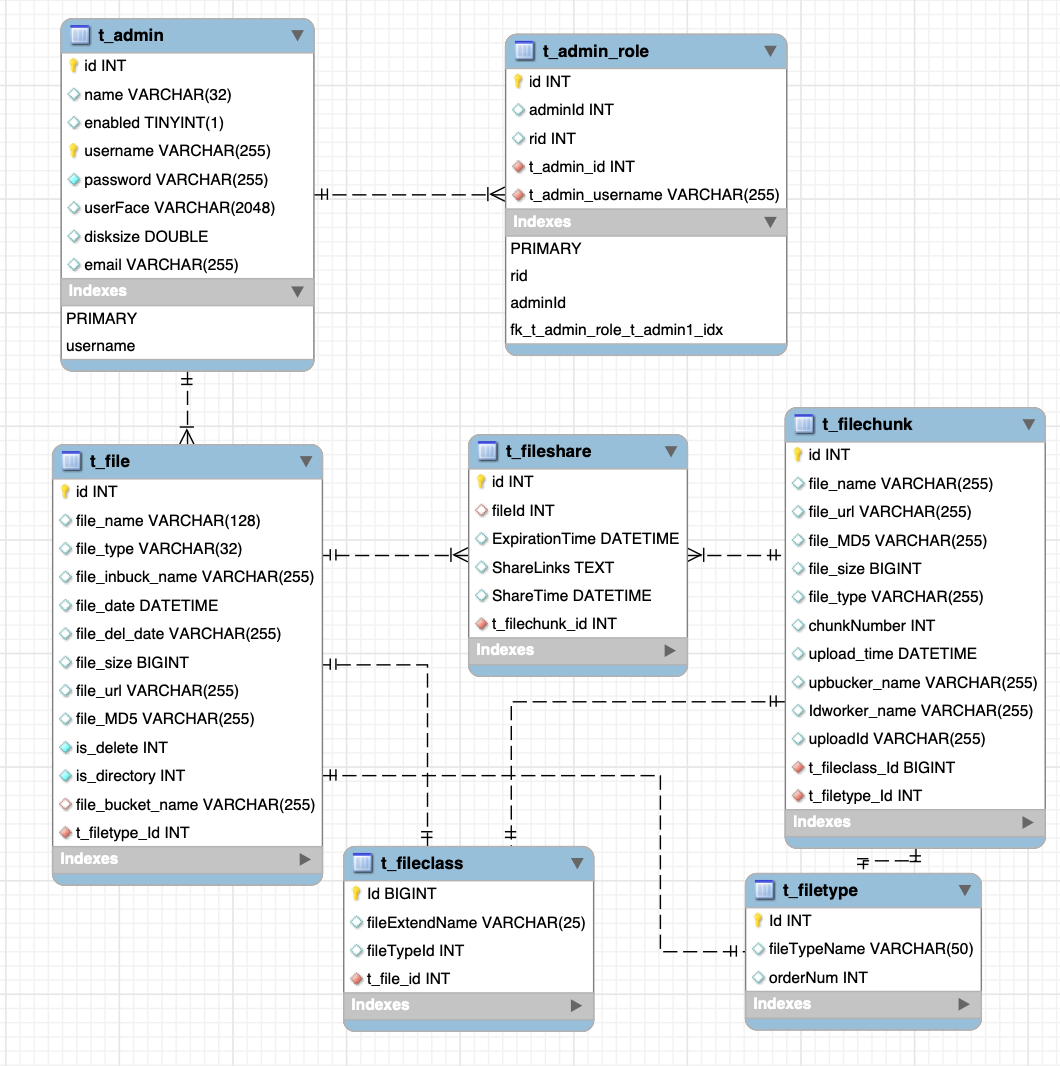


Figure 5.7 File ER Detail Diagram

The following screenshot shows the actual implementation of the system, which are 5.8 File List, 5.9 File Upload(in process status), 5.10 File Upload(done status), and 5.11 Big File Chunk Upload：

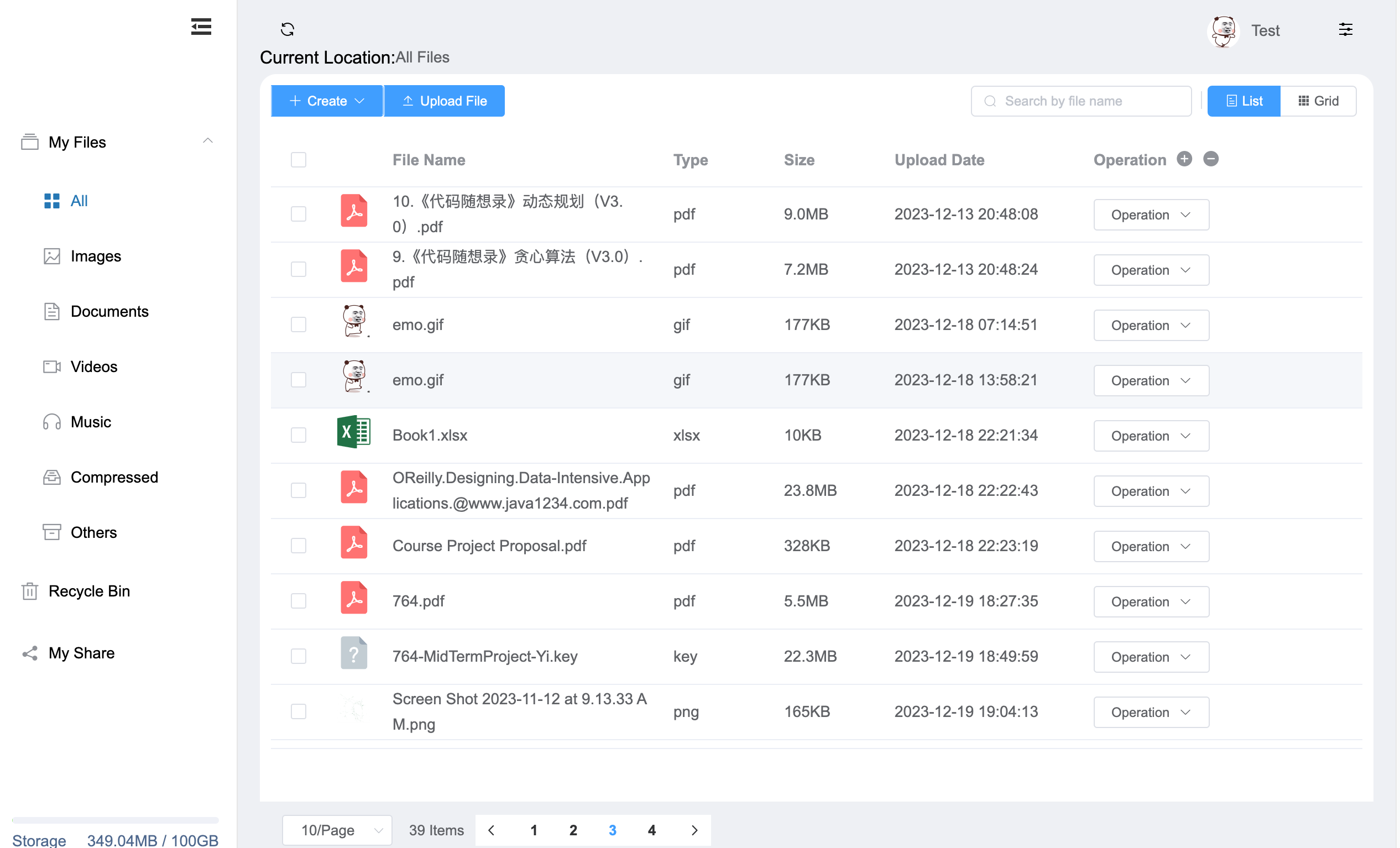


Figure 5.8 File List

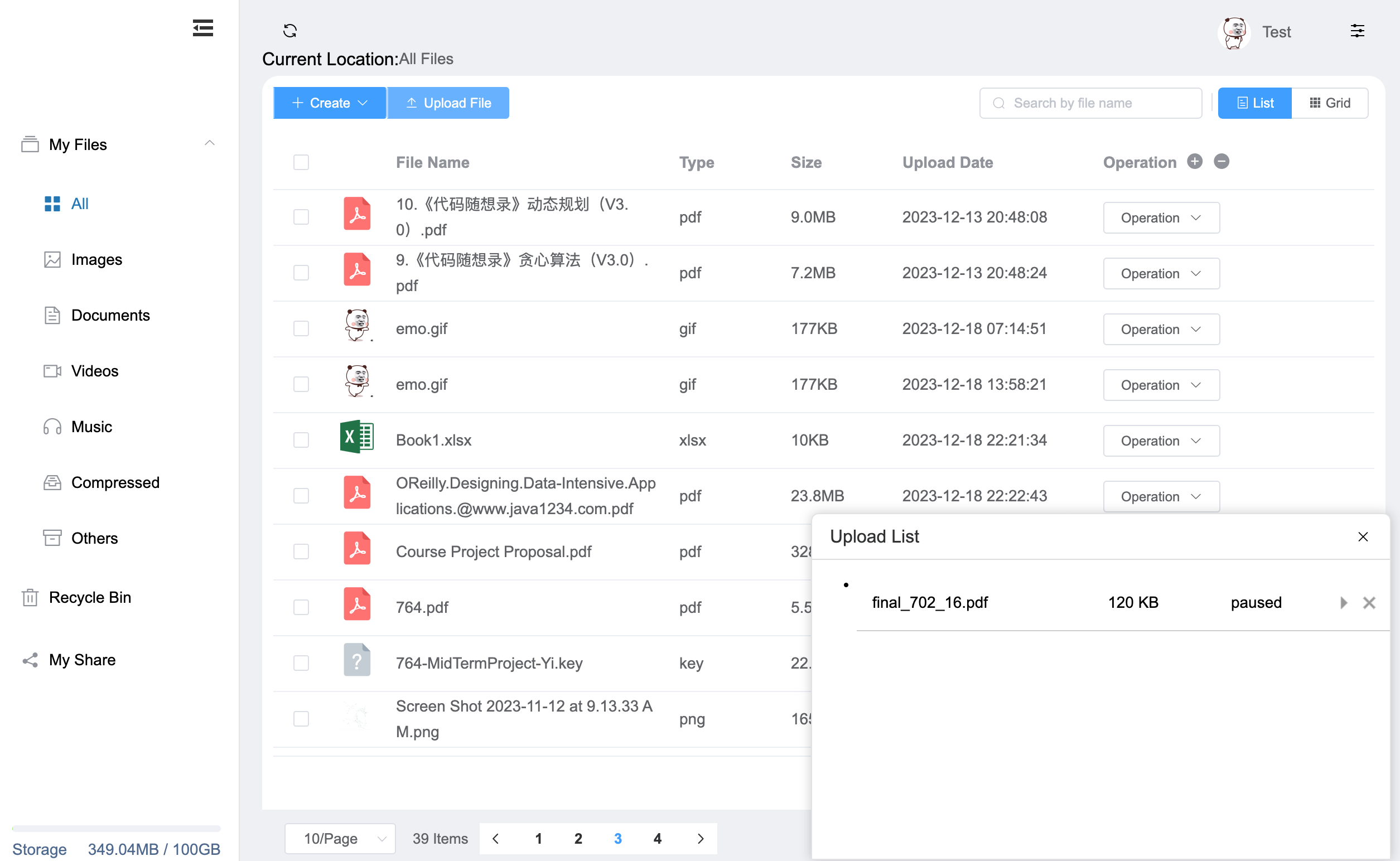


Figure 5.9 File Upload(in process status)

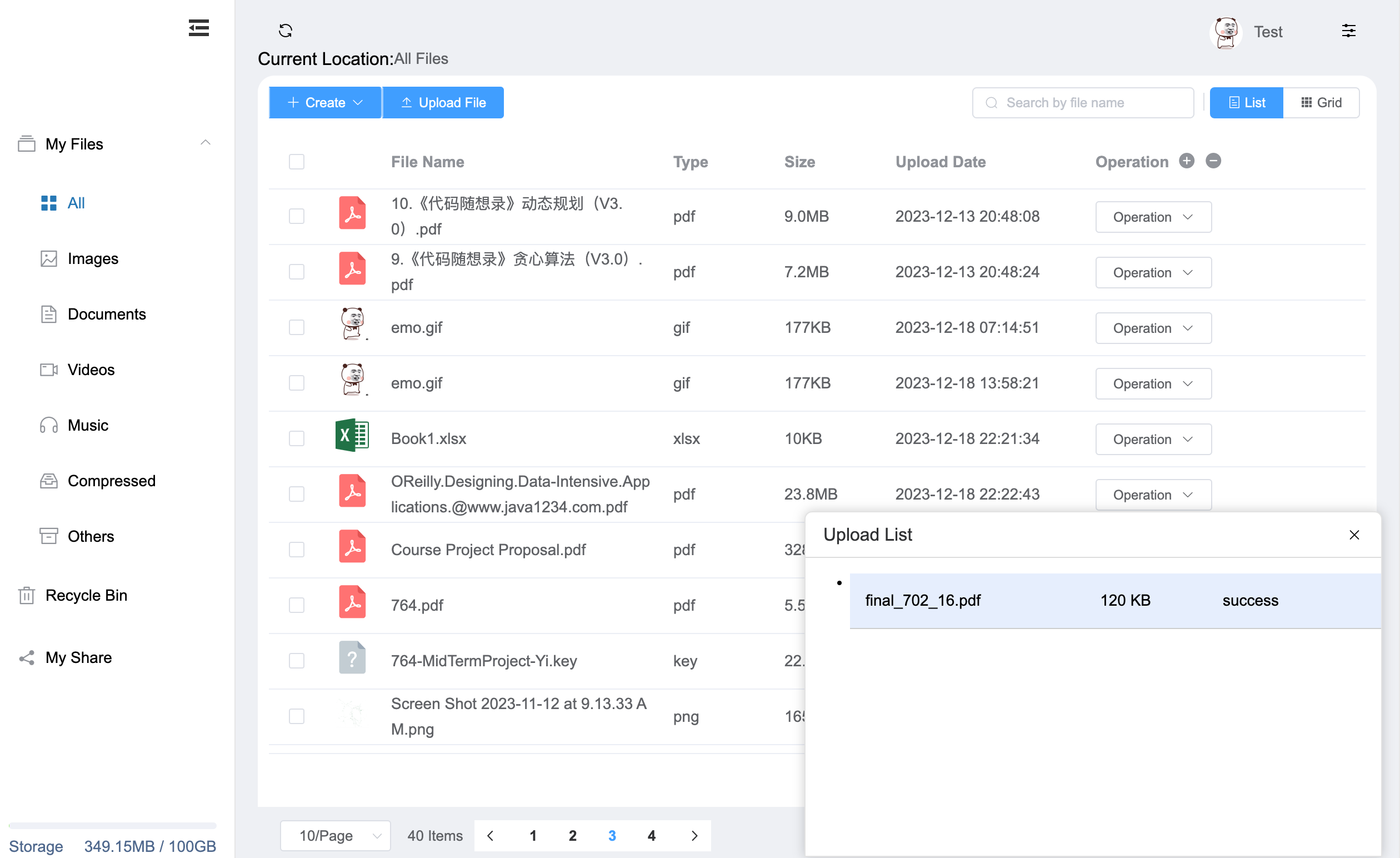


Figure 5.10 File Upload(done status)

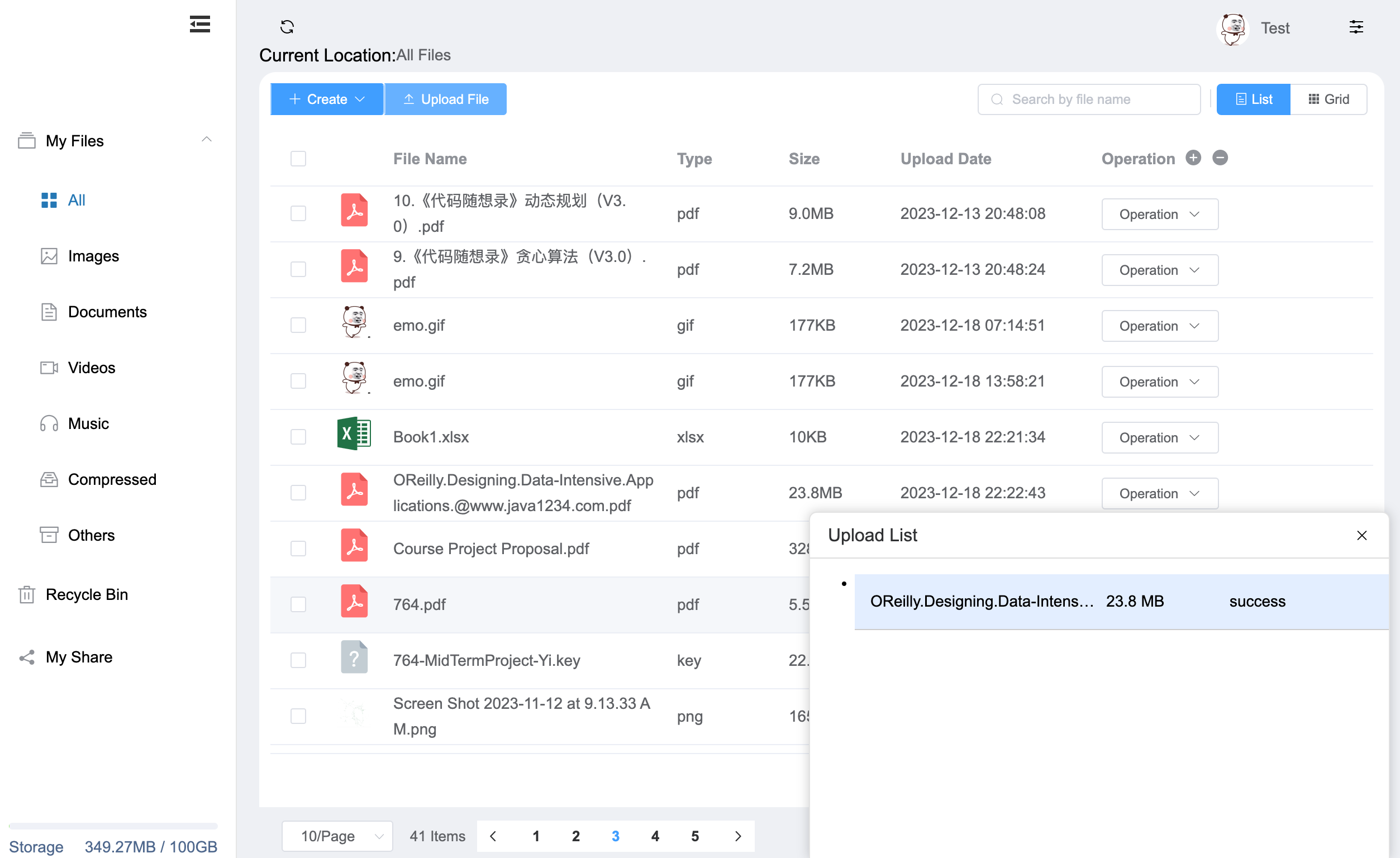


Figure 5.11Big File Chunk Upload

## 5.2.3 Preview Domain

In the pre-demonstration section, the vue component decides which component to call based on different file types with bound file extensions. When a user clicks on a file in the file management interface, the system first retrieves the file's MIME type or extension and passes this information to a central scheduler. Based on this information, the scheduler dynamically loads the corresponding Vue component.

If the file is an audio type such as .mp3 or .flac, the scheduler calls an audio player component with embedded HTML5 <audio> tags and playback controls such as play, pause, and adjust volume. These controllers are bound to Vue's data model, enabling a responsive user interface. Similarly, for image files such as .jpg or .png, the scheduler then loads an image display component, which may use <img> tags and comes with some additional functionality such as zoom and swipe to view.

On a technical level, these components are implemented as Single File Components for Vue, an architecture that allows each component to have independent scopes and lifecycles, which helps with maintenance and code reuse. Component communication occurs through Vuex managed state or through parent-child component props and events. Such modular design approach improves development efficiency and ensures a highly scalable and maintainable front-end interface. With these well-designed components, the web hosting system can provide users with one-stop file preview and playback experience, which greatly improves the ease of use and practicality of the web hosting system.

The following screenshot shows the actual implementation of the system, which are 5.12 Music Preview 5.13Picture Preview：

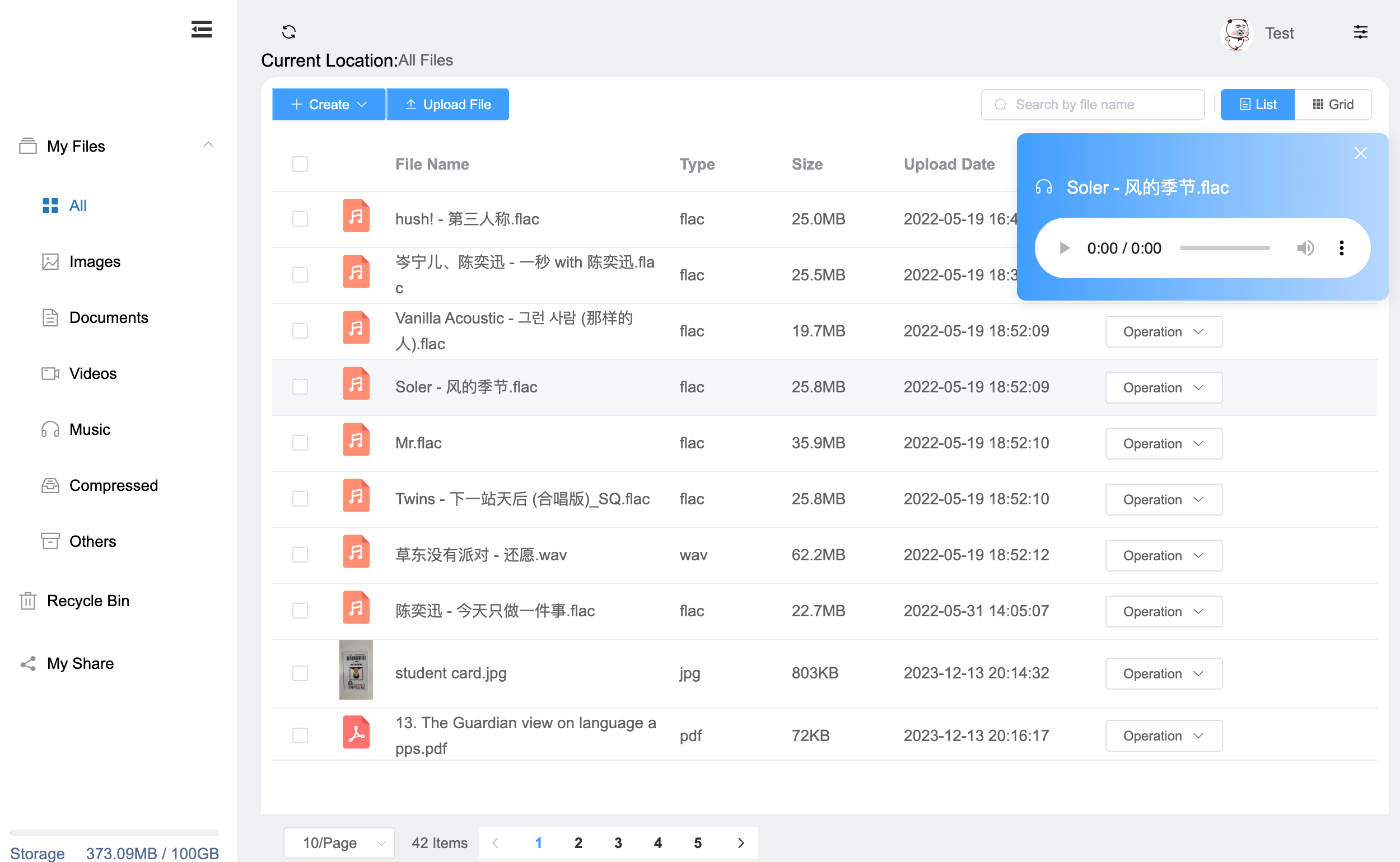


Figure 5.12 Music Preview

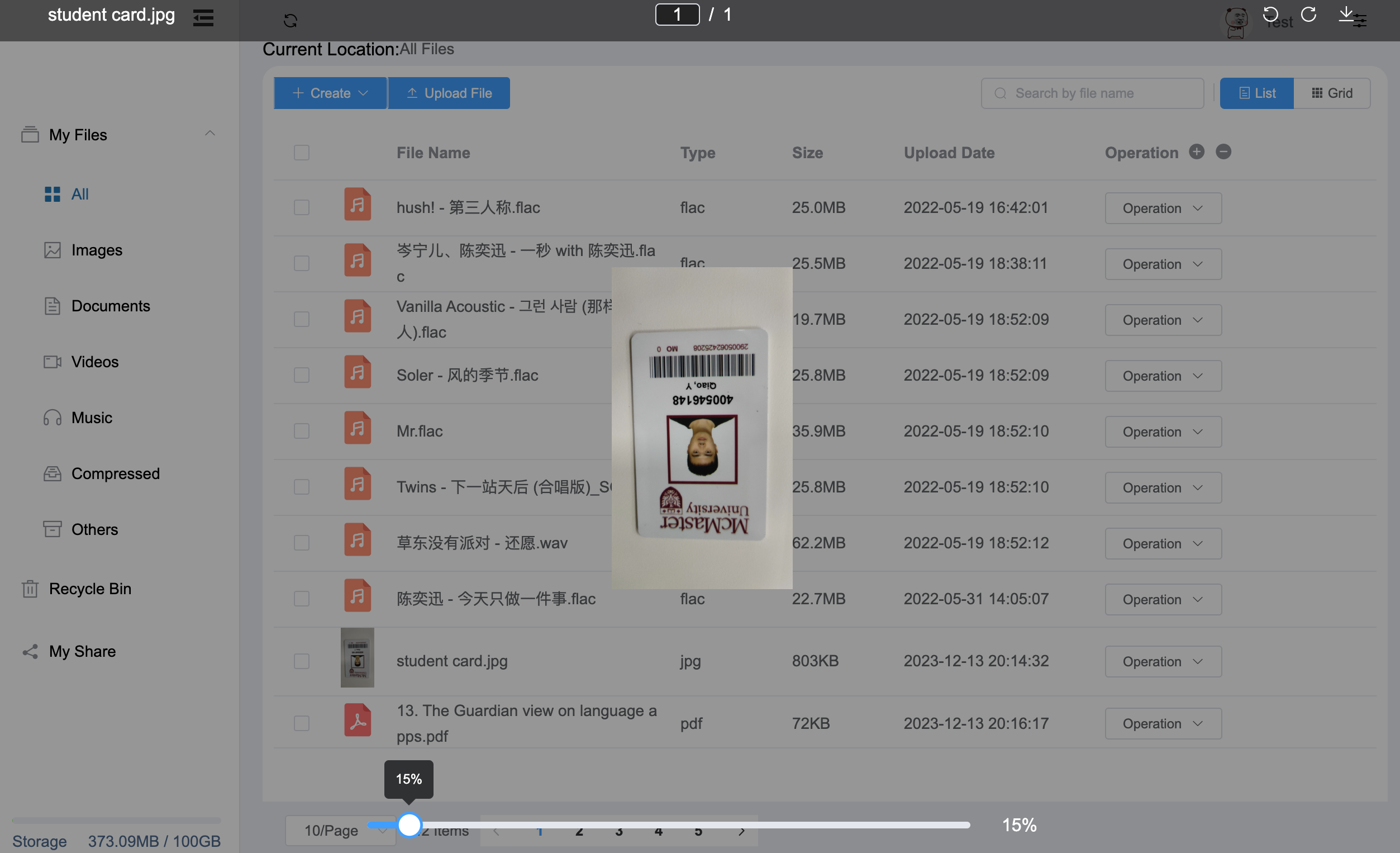


Figure 5.13 Picture Preview

# 5.2 Interface Development



Figure 5.14 Back-end Interface Display with Swagger

<http://localhost:8081/doc.html?cache=1&lang=en>

# EVALUATION

As the code base expands, small errors and unexpected edge cases can lead to larger failures. Errors can lead to a poor user experience and ultimately a loss of business. One way to prevent vulnerable programming is to test the code before releasing it. When problems are found at this stage, tasks are reassigned to developers in order to fix them.

Unit tests are a core part of modern code; they allow for completely isolated and separate inspection of properties in a program, helping to detect errors during development. Automated testing frameworks (e.g., xUnitcollection) have made writing unit tests easier and more convenient than ever. However, writing high-quality unit tests is still not a trivial task[[22]](#endnote-21)[19]. In this thesis, I will focus on testing the back-end service, as the front-end service is not my core service, and the whole project will be more oriented towards the back-end model design and implementation. Therefore, I will use JUnit, which is a common industry testing method, to realize the unit testing of back-end services.

## 6.1 Writing Test Cases

In the process of software development, writing testable code is one of the key steps in ensuring the quality of the final product. In this study, we pay special attention to writing clear, modular, and maintainable code for system testing. In Spring Boot back-end development, we follow the single responsibility principle to ensure that each method and class has a clear and independent function, which not only facilitates code readability and maintainability but also simplifies the testing process.

To improve the testability of our code, we adopted the Dependency Injection (DI) design pattern, which allows us to replace or simulate external dependencies in the test environment, thus enabling more accurate unit testing. Additionally, we leveraged Spring Boot's built-in testing frameworks, which provide a rich set of test annotations and tools that allowed us to automate the testing of the RESTful APIs to ensure correct interface behavior.

In Vue.js front-end development, we ensure the testability of our code through a componentized structure. Each component is designed as a standalone unit that can be tested without relying on other components. By using tools such as Vue Test Utils, we are able to simulate user interactions and ensure that components work in different situations.

## 6.2 Writing and Structuring Testing

Testing is not just a phase of the code development process, but an ongoing activity throughout the software lifecycle. We adopted a test-driven development (TDD) approach, where test cases are first written and then functionality is implemented to pass the tests. This approach not only exposes potential problems in advance, but also promotes better design decisions. In Spring Boot, we use JUnit and Mockito frameworks to write and execute unit tests that validate the behavior of methods by simulating dependencies.

Integration tests are another important part of our testing strategy, ensuring that interactions between different modules work as expected. We use Spring Boot's testing framework to perform integration tests that cover the complete flow from database operations to RESTful API requests. For the front-end Vue.js, we utilized the Karma and Jest frameworks to perform unit and end-to-end (E2E) testing to ensure that the interaction logic of the user interface is correct.

## 6.3 Unit Testing

## 6.4 End-to-End Testing

## 6.5 Requirements Coverage

# CONCLUSION

In this paper, we have successfully designed and implemented a web storage system based on MinIO, Spring Boot, and Vue.js. By fully utilizing the advantages of MinIO in the field of object storage, combining the efficient back-end development capability of Spring Boot and the dynamic front-end interactivity of Vue.js, I explored an efficient and user-friendly cloud storage solution. The system architecture uses the DDD model design approach for model design and development based on the model domain. During the implementation of the system, we also encountered various technical challenges, including how to design the model and how to ensure data interaction. Moreover, since I was previously a specialized back-end programmer and did not have much involvement in front-end technologies, the solutions to these challenges not only enhanced the technical depth of our system.

## 7.1 Future Work

I hope to continue exploring and expanding the functionality of the netdisk system based on the existing foundation. First of all, the research of the second transfer function will be one of the focuses of future work. The second transfer can significantly improve the efficiency of large file transfers. By hashing files, repeated uploading of files with the same content is avoided, thus saving transmission time and storage space. Secondly, the research on encrypted transmission will also be included in the future development program. With the increasing threats to network security, it is becoming more and more important to ensure that data is not stolen or tampered with during transmission. Therefore, research on how to incorporate stronger encryption mechanisms in the transmission layer will further enhance the security of the system.

In addition, the application of WebSocket protocol is also an important direction for future development. We currently use the HTTP protocol for development, which is of course the most mainstream development scheme, but WebSocket provides a way of full-duplex communication over a single TCP connection, which is conducive to real-time data transfer and faster communication response. In the netdisk system, more instant notification of file updates and smoother multi-user collaboration can be realized by using the WebSocket protocol. In summary, by introducing the second transfer function, strengthening the encrypted transmission, and utilizing the WebSocket protocol, I believe that we can greatly improve the performance and user experience of the netdisk system, and at the same time contribute to the development of cloud storage technology. Future work will continue to work in these directions to meet the growing needs of users and to address new challenges in the field of data storage and transmission.

# REFERENCES

1. [1] Mell, P., & Grance, T. (2011). The NIST definition of cloud computing. [↑](#endnote-ref-0)
2. [2] Armbrust, M., et al. (2010). A view of cloud computing. Communications of the ACM, 53(4), 50-58. [↑](#endnote-ref-1)
3. [3] Bello, S. A., Oyedele, L. O., Akinade, O. O., Bilal, M., Davila Delgado, J. M., Akanbi, L. A., Ajayi, A. O., & Owolabi, H. A. (2021). Cloud computing in construction industry: Use cases, benefits, and challenges. Big Data Enterprise and Artificial Intelligence Lab (Big-DEAL), University of the West of England (UWE), Bristol, United Kingdom. Available online 17 December 2020, Version of Record 1 January 2021. [↑](#endnote-ref-2)
4. [4] Amazon Web Services, Inc. (2021). "Amazon Simple Storage Service (Amazon S3) documentation." [↑](#endnote-ref-3)
5. [5] Microsoft. (2021). "Azure Blob Storage documentation." [↑](#endnote-ref-4)
6. [6] Google Cloud. (2021). "Google Cloud Storage documentation." [↑](#endnote-ref-5)
7. [7] Weil, S. A., et al. (2006). "Ceph: A scalable, high-performance distributed file system." In Proceedings of the 7th symposium on Operating systems design and implementation. [↑](#endnote-ref-6)
8. [8] The Rook Authors. (2021). "Rook: Open-source, cloud-native storage for Kubernetes." [↑](#endnote-ref-7)
9. [9] Buuck, B. (2023, December 12). A Guide to Decoupled Storage with StarRocks and MinIO. MinIO Blog. Retrieved from https://blog.min.io/tag/open-source/ [↑](#endnote-ref-8)
10. [10] MinIO. (n.d.). Erasure Coding. MinIO Object Storage for Linux. Retrieved [the date you accessed the website], from https://min.io/docs/minio/linux/operations/concepts/erasure-coding.html [↑](#endnote-ref-9)
11. [11] Auernhammer, A. (2022, December 16). MinIO Best Practices - Security and Access Control. MinIO Blog. Retrieved from https://blog.min.io/s3-security-access-control/

    [12] You, E. (2014). "Vue.js: A progressive, incrementally-adoptable JavaScript framework for building UI on the web." [↑](#endnote-ref-10)
12. [13] DoNotApply(2022, Aug, 17) Vue vs Angular, which has the steeper learning.Medium. <https://medium.com/@donotapply/vue-vs-angular-which-has-the-steeper-learning-curve-b75df9ab9a0a>

    [14] Vernon, V. (2013). Implementing Domain-Driven Design. Addison-Wesley Professional.

    [15] Evans, E. (2013). Getting Started With DDD When Surrounded By Legacy Systems. Domain Language, Inc.

    [16] Evans, E. (2003). Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Professional.

    [17] Doe, J. (2021). kaptcha. Retrieved from https://code.google.com/archive/p/kaptcha/

    [18] Kavitha, C. R., & Thomas, S. M. (2011). Requirement gathering for small projects using agile methods. IJCA Special Issue on Computational Science-New Dimensions & Perspectives, NCCSE.

    [19] Moritz Beller, Georgios Gousios, Annibale Panichella, and Andy Zaidman. 2015.When, How, and Why Developers (Do Not) Test in Their IDEs. In Proceedingsof the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE2015). ACM, New York, NY, USA, Article 1, 12 pages.

    [20] Khononov, V. (2021, October). Learning Domain-Driven Design: Aligning Software Architecture and Business Strategy. O'Reilly Media. [↑](#endnote-ref-11)
13. [↑](#endnote-ref-12)
14. [↑](#endnote-ref-13)
15. [↑](#endnote-ref-14)
16. [↑](#endnote-ref-15)
17. [↑](#endnote-ref-16)
18. [↑](#endnote-ref-17)
19. [↑](#endnote-ref-18)
20. [↑](#endnote-ref-19)
21. [↑](#endnote-ref-20)
22. [↑](#endnote-ref-21)