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](#_Toc1992971544)

[1. INTRODUCTION 5](#_Toc1570297749)

[1.1 Background and Significance of the Study 5](#_Toc1567729460)

[1.2 The Need for Research 5](#_Toc1352169177)

[1.3 Research Content and Objectives 5](#_Toc1235405285)

[1.4 Research Methodology and Technical Route 6](#_Toc1584725799)

[1.5 Organization of the paper 6](#_Toc1394313699)

[2. RELATED WORK 8](#_Toc888783029)

[2.1 Overview of Cloud Storage Technology 8](#_Toc2027603518)

[2.2 Existing Cloud Storage Services 8](#_Toc1661816430)

[2.3 Open-Source Cloud Storage Solutions 9](#_Toc2123909775)

[2.4 MinIO: High Performance, Kubernetes-Native Object Storage 9](#_Toc1078407991)

[2.5 Vue.js: An Incremental JavaScript Framework 10](#_Toc41124057)

[3. REQUIREMENTS 12](#_Toc1829775312)

[3.1 Requirements Gathering 12](#_Toc1067843744)

[3.2 Functional Requirements 12](#_Toc728967429)

[3.3 Non-functional Requirements 13](#_Toc361373068)

[4. DESIGN 15](#_Toc513400160)

[4.1 Overall Design 15](#_Toc127195474)

[4.2 Domain Detail Design 17](#_Toc1028102753)

[4.2.1 User and Auth Domain 17](#_Toc669545909)

[4.2.2 File Domain 18](#_Toc243782283)

[4.2.3 Preview Domain 21](#_Toc1997515552)

[5. IMPLEMENTATION 23](#_Toc632028913)

[5.1 DDD Development 23](#_Toc1055822729)

[5.1.1 Code Structure 23](#_Toc555231142)

[5.1.2 Overall ER Diagram 24](#_Toc953357379)

[5.2 Functional Implement 25](#_Toc701978586)

[5.2.1 User Domain 25](#_Toc2026421931)

[5.2.2 File Domain 27](#_Toc1130236544)

[5.2.3 Preview Domain 29](#_Toc1392737293)

[5.2 Interface Development 30](#_Toc163931151)

[6. EVALUATION 32](#_Toc2116819403)

[6.1 Writing Test Cases 32](#_Toc22126372)

[6.2 Writing and Structuring Testing 32](#_Toc363263273)

[6.3 Unit Testing 33](#_Toc69820890)

[6.4 End-to-End Testing 34](#_Toc953626968)

[6.5 Performance Testing 35](#_Toc937993615)

[7. CONCLUSION 36](#_Toc181234678)

[7.1 Future Work 36](#_Toc879421700)

[A. REFERENCES 37](#_Toc1458053246)

# INTRODUCTION

## 1.1 Background and Significance of the Study

In the present era of rapid development of information technology, data and information have become the lifeblood of social operation. How to store and access these data efficiently and securely as well as how to ensure the isolation of private data has become very important. 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, represented by Minio, provide users with options to customize their services to adapt to changing storage needs and challenges. This study focuses on the application of open source cloud storage services and explores the design and implementation of a Minio-based cloud disk application, aiming to provide users with high-performance, highly reliable, and easy-to-manage data storage options.

## 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 the distributed architecture, which allows data to be stored in several locations, thereby increasing redundancy and reliability. Due to this speedy growth, these data storage clouds have gained a meaningful status in the data management strategy. 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]. Cloud storage technology is the new standard that enables better UX consistency and speed among regular storage solutions. This is why it has been widely adopted. Furthermore, the ability to acquire data at any time and from any place has technologically modified business activities, which in turn created another field of information interchange and collaboration among people elsewhere. The role of such developments in technology is unquestionable, since they determine the next generation of the Internet or services and applications.

## 2.2 Existing Cloud Storage Service

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].

Though these providers are becoming increasingly powerful, there are a few problems. The cloud service lock-in issue is one of the biggest problems, since each of the unique hierarchical human functions and API interfaces offered by the different services make it almost impossible for clients to move their data to others. To add to this, choosing data transfer and operations and analyzing their aims can be challenging, and sometimes even impossible. Additionally, this problem is also a challenge due to the emergence of new data protection legislation, which requires cloud providers to comply constantly with the legal regulations. This fact constitutes the reason why cooperation between the industry innovation and the compliance at the same time with the cost reduction is the main goal of the cloud storage for now.

## 2.3 Open-Source Cloud Storage Solutions

In line with the upsurge in open-source cloud storage solutions, the market has experienced positive revitalization, characterized by offerings that put an emphasis on privacy, adaptability, and crowd-driven development. These resolutions apply to a large plethora of issues, such as individual, cloud storage, or business settings. An illustration of this is the Ceph storage system, which is a uniform and distributed storage system fully equipped with high efficiency and elasticity. 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 further strains for development of novel cloud storage technology, thus adding value. Users can roll out and tweak their storage equipment to fit specific functions even if it means moving beyond the traditional proprietary services. Besides the community development methodology, which has fast development cycles, new concepts, like the erasure coding and geo-replication, ensure better durability and availability of data. Apart from this, ability to look at the source code and modify it also acts as a layer of security and trust, since any vulnerability within the system can be reported to the developer community and thus rectified quickly.

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

Even though MinIO might be a newcomer in the domain, it is proving to be a leading technology that is often selected by many businesses due to its supreme features and compatibility with S3 APIs, which are often chosen for cloud storage solutions. Initiated from the primary focus of private cloud and containerized environments, MinIO is the right choice for a highly scalable platform that can accommodate a variety of data-driven applications, starting from machine learning and extending 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 the safety of data can be termed as praiseworthy. The service includes a set of security measures, for instance, end-to-end encryption, impersonation detection, and access control management, as well as the creation of very detailed 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 is a JavaScript framework that provides a platform for building user interfaces, which is using a model we find more efficient than others. As distinct from other frameworks that use a monolithic approach, Vue was designed for gradual implementation from the day it was born. The core library, the working world, caters to the view only for the project, which makes it easy to incorporate with other libraries or with projects.

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.[[13]](#endnote-12)[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[[14]](#endnote-13)[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 precise research on core features, interface design, development model, and after sales services of these products, we could theoretically appropriate their techniques and be able to discover the industrial best practices for us. The book, for instance, Amazon Cloud Drive is a great drift towards collaboration, Baidu Cloud Drive is the one that possesses the high percentage of local customers. And so, after having this information, we will be in a position to examine the demand and pain points of our target users for needed decisions and methodologies to differentiate our Cloud Drive product from the competitors.

## 3.2 Functional Requirements

The part which the user is coming to contact is the registration process, thus we must set a less difficult registration process, but at the same time strong enough to avoid temptation from hack. Users can register via either an email ID or a mobile number and have it verified using an unique code. After registration, this person gets logged in, but there should be a number of safety-checking triggers, for example, prohibition and warning prompts whenever a user enters wrong password. We can also implement a social media login to solve this problem by the convenience of users.

User Authentication (Forensics): The other condition is that the system provider must ensure that only registered users get access to the resources that they have permission to access. This shall be done by using forensic devices that allow verification of the user's identity before such operations as password change, access to personal files, and so on are executed. The implementation of such system may occur using OAuth 2.0 or JWT, which provide secure management of users' sessions and protect the safety of the executions between servers.

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 one of the most powerful mechanisms that allow for better upload of large files and improves efficiency and stability. Instead of sending one huge file, that is risky and which may fail to upload, data is divided into several smaller chunks for better efficiency in network bandwidth use and redundant transmission. Not only being in terms of data transfer, it improves the security with encryption of separate chunks, which provides additional benefits too. Therefore proper care must be taken in this technique, to make sure other factors such as the slice size, recovery mechanisms of data after transmission drops, and how to organize the slice on the server side are effectively looked into.

While examining the file search feature, efficiency and precision remain the central components of this investigation. To ensure quick and precise file lookup, a systematic validation mechanism should be crafted, and this system may need to integrate large file attributes and content indexes. We will be looking at complex search algorithms and user queries. Although they may involve complex search algorithms and user interface design, these systems should return results quickly even in very large datasets. This can be achieved through various criteria like file name, type, or modification date. A searching function must be designed based on users' experiences, and scalability and performance of the backend should cope with data volume.

Distributed Minio, the basis of cloud storage, constructs a system with high levels of redundancy and availability. Minio's technique of data distribution across different server nodes resembles the process of decentralized storing data, therefore even if some nodes fail, the data will store on safe and the service will remain sustainable. This means that we can use efficient data syncing technologies to achieve it in a way that keeps data up-to-date 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.[[15]](#endnote-14)[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[[16]](#endnote-15)[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[[17]](#endnote-16)[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:

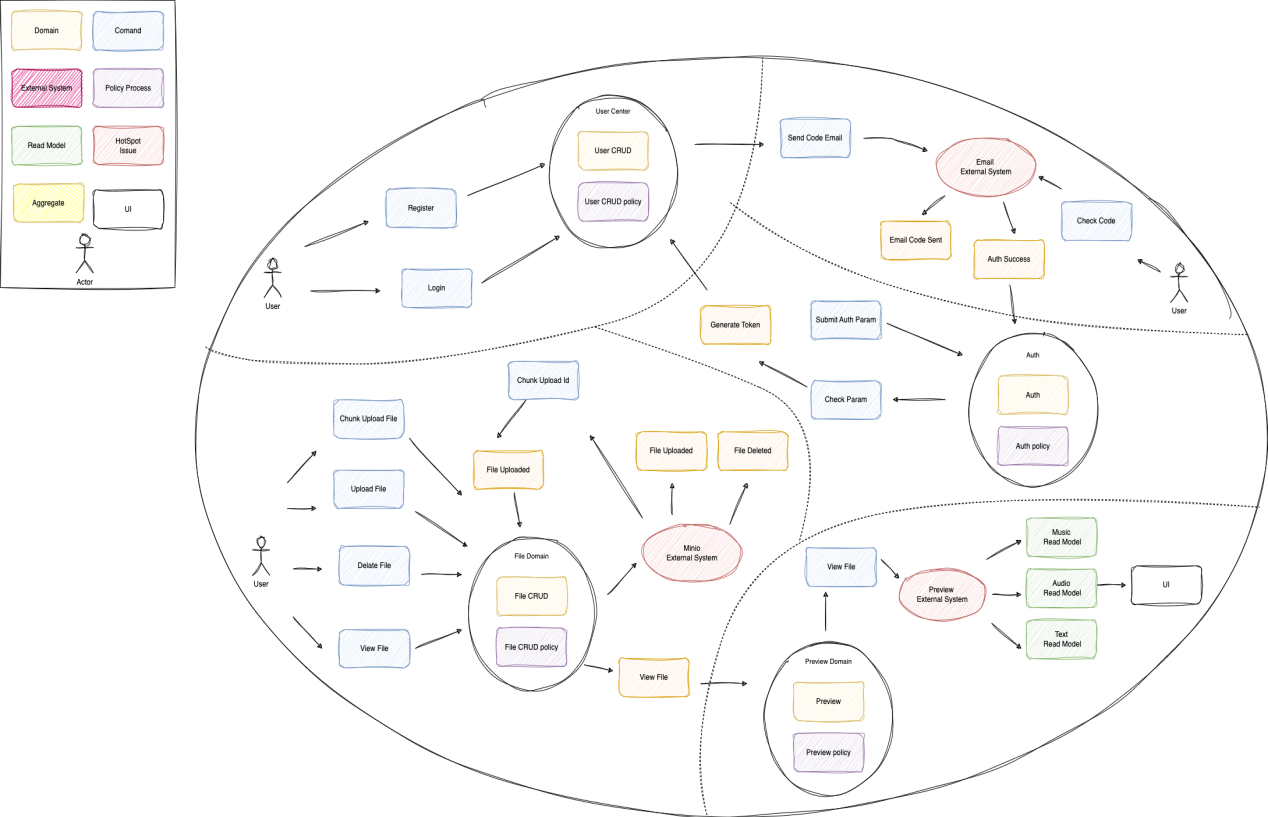


Figure 4.1 Overall DDD Entity and Context Design

## 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.

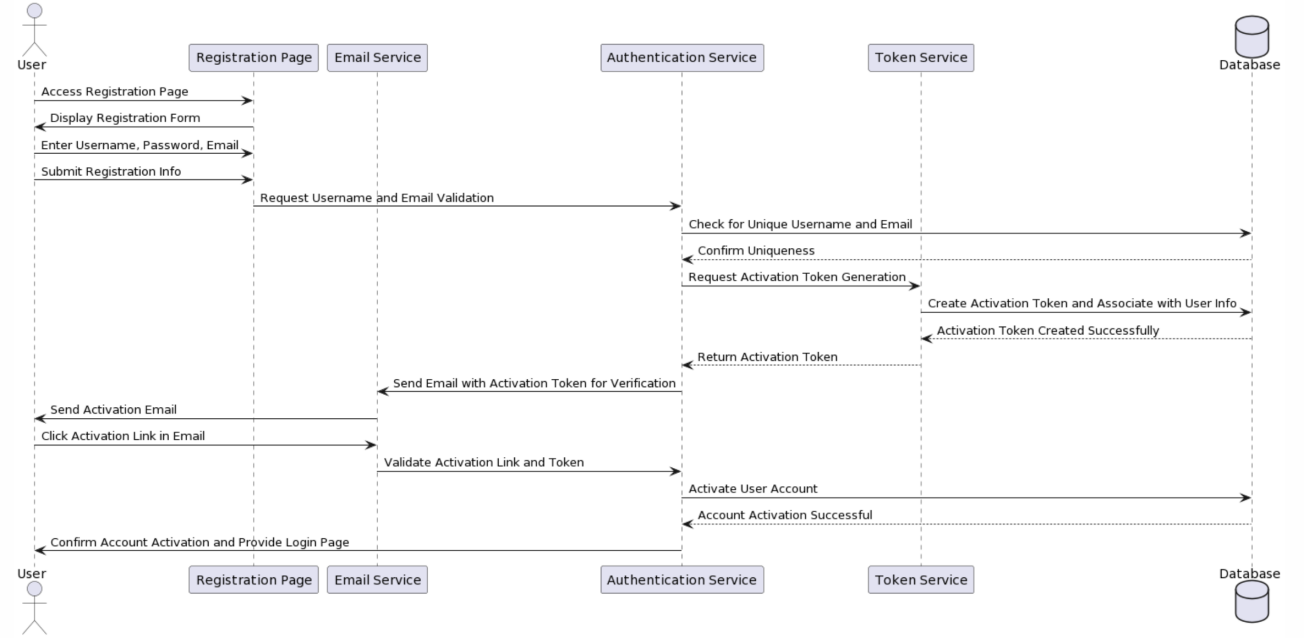


Figure 4.2 User Register Sequence Diagram

#### 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[[18]](#endnote-17)[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.

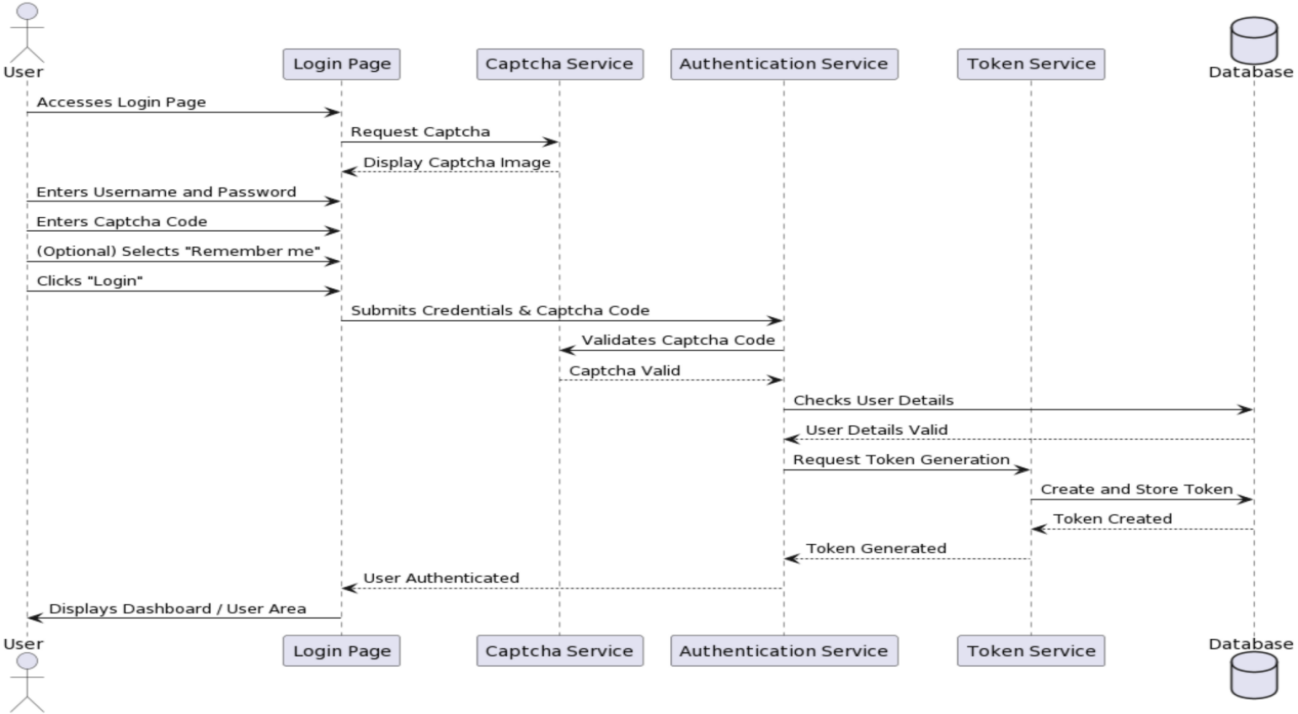


Figure 4.3 User Login Sequence Diagram

### 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.

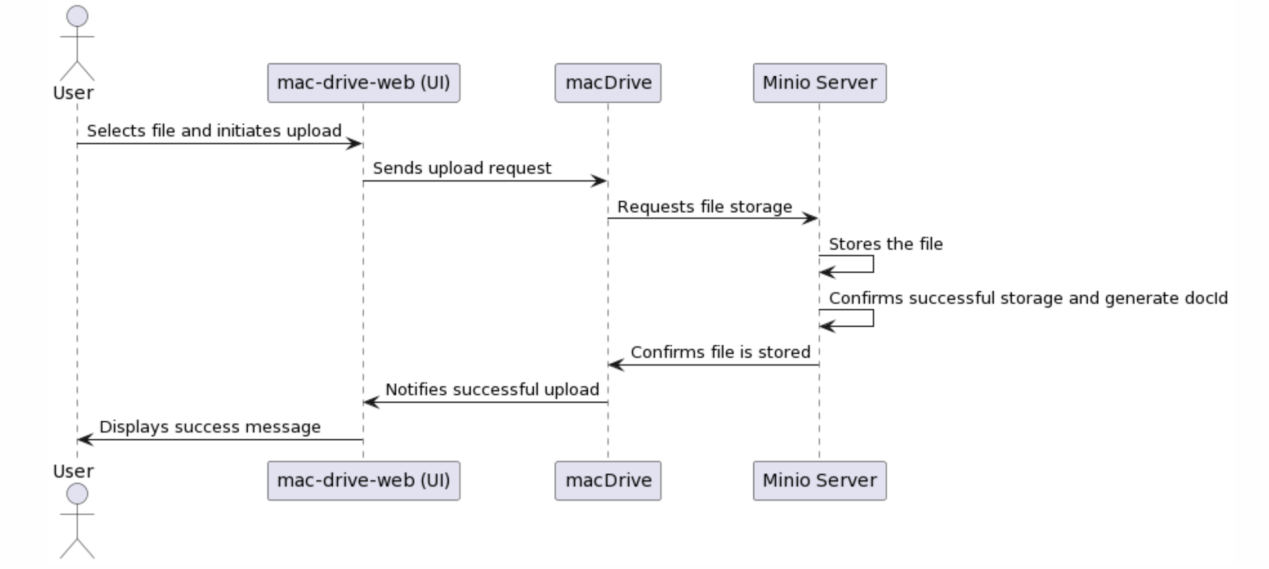


Figure 4.4 User Upload File Sequence Diagram

#### 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. After the upload of all chunks being successful, the MinIO server ensures the proper reassembly of chunks into a single file and returns as the response the reassembled file metadata to the MinIO server. The MinIO server saves the information regarding the metadata of files and in turn, we can find out that metadata has been saved 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. So there is a process, and in the end, macDrive-web shows that there was a file uploaded successfully.

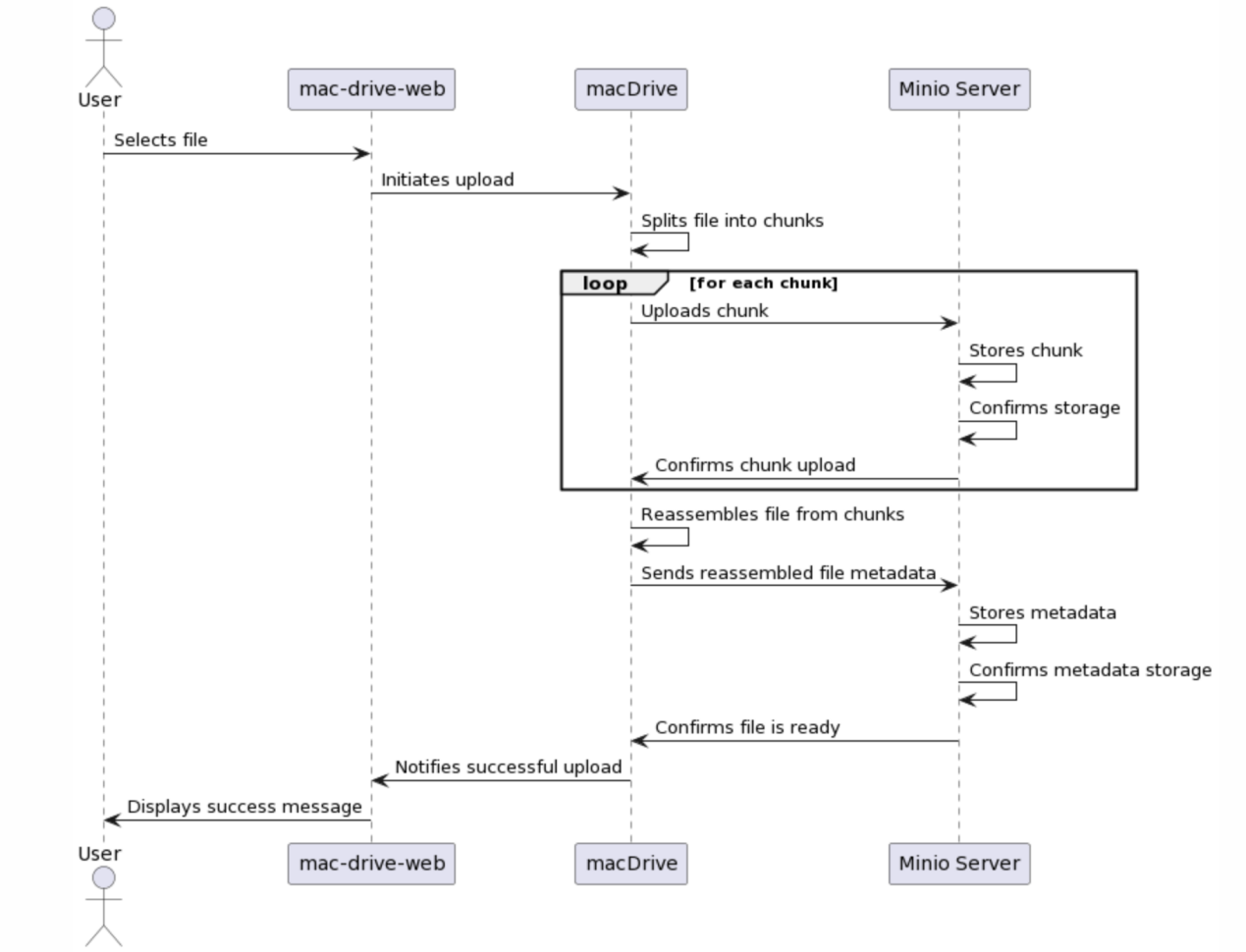
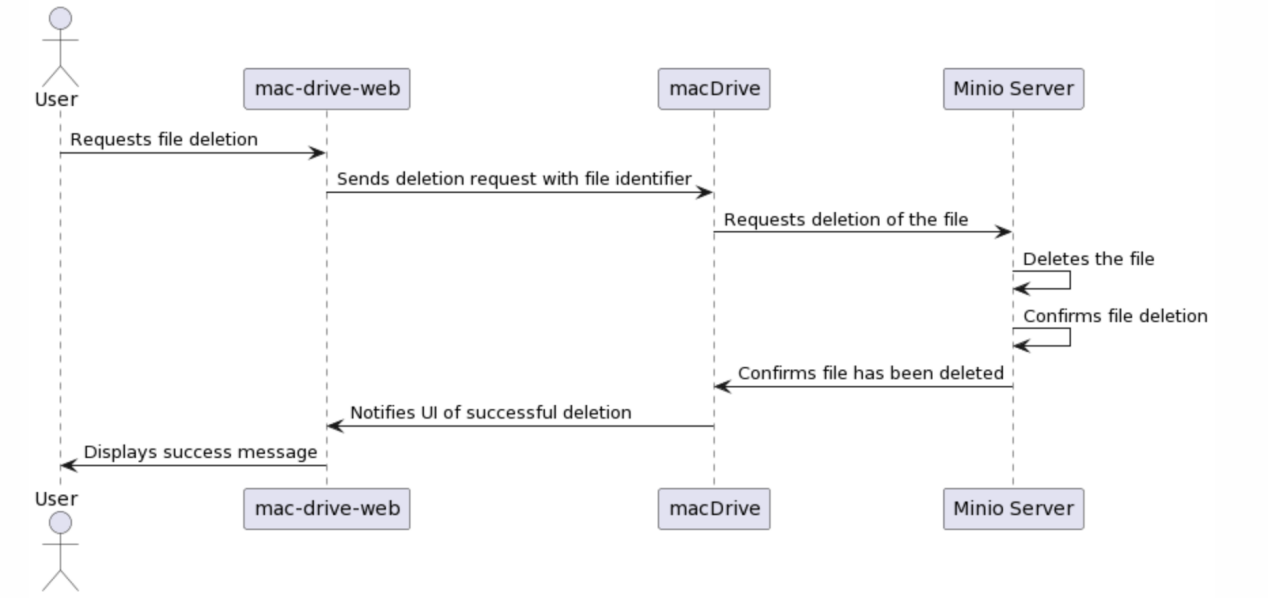


Figure 4.5 User Chunk Upload File Sequence Diagram

#### 4.2.2.3 File Delete

The diagram of process, the user deletes a file from the web interface, is in the sequence. The system user mentions the file to be deleted from the mac-drive-web interface. This identifier designates the requested file. This applies through macDrive by sending a delete command to the Minio server, which issues a confirmatory response to 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 diagram is a sequential one and marks the user putting a request in order to look through the files on the mac-drive-web interface. Do this, the mac-drive-web server starts a communication with the macDrive, which then shows the files on the web browser. 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 server creates a list of files it has found and returns this to macDrive, thereby providing macDrive with a list of files. This return of files from macDrive will be back to the macDrive interface page. The macDrive will again send the file-list back to the mac-drive-web interface in which case the user will finally be shown the list of files.

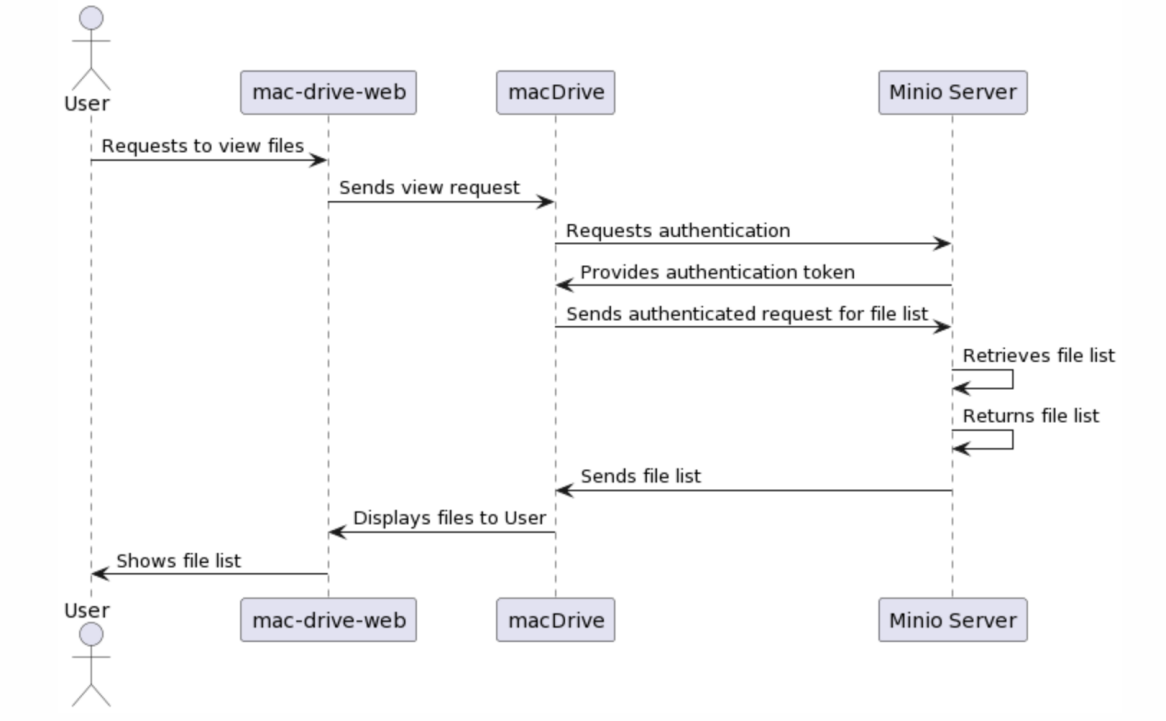


Figure 4.7 File Display Sequence Diagram

### 4.2.3 Preview Domain

Here, the sequence diagram is showing that the user starts the process by using the mac-drive-web and sends the file request to the OS for those files to be viewed. mac-drive-web forwards this request to macDrive, which requests that the user authenticate. The server receives an authenticated request from macDrive after authentication has been granted permission to the user to get the file list from the Minio server for the granted access rights. Here, the Minio server handles this request, puts the files into a list, and sends it to macDrive. macDrive asks another service to handle the file types which processes the files it finds in the list as either music, video, or document. When this service finishes processing, it comes back and returns this information to macDrive. The macDrive portion obtains the file type information and then sends the file and its group back to the mac-drive-web. This last code segment displays the name of the files with their corresponding icon, thereby assisting the user to identify files visually and facilitates them to browse through the files according to the file type.

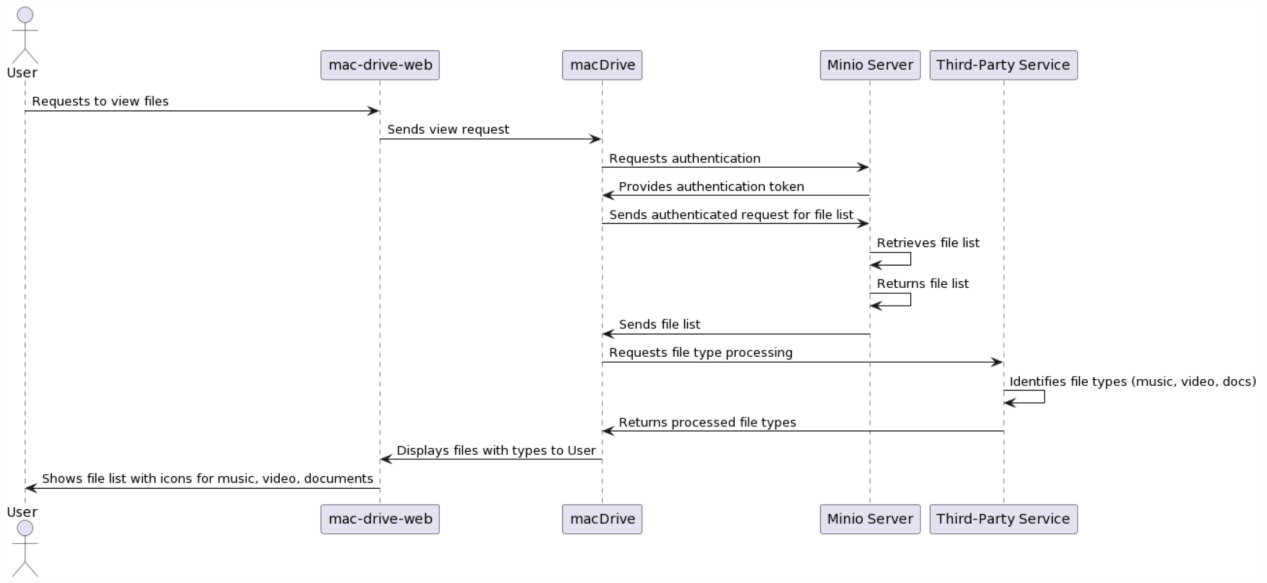


Figure 4.8 Preview Domain Sequence Diagram

# 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[[19]](#endnote-18)[20].

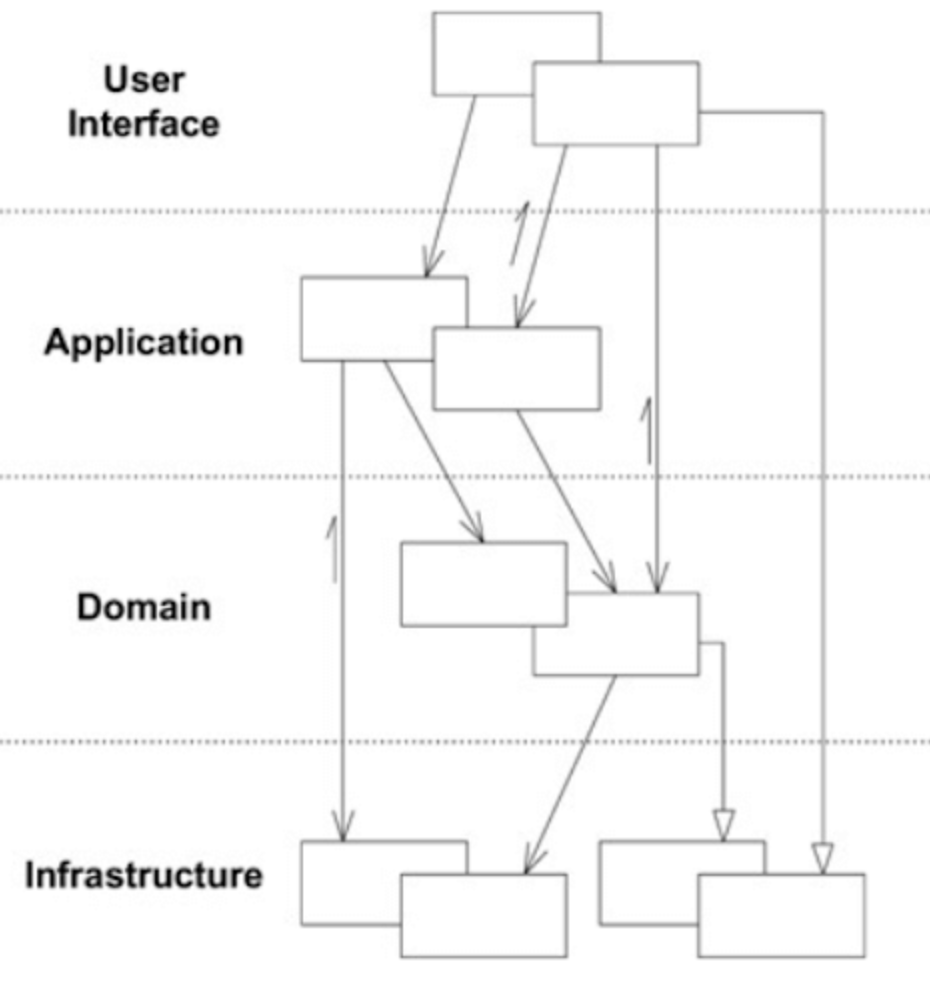


Figure 5.1 DDD Layer[[20]](#endnote-19)[20]

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；**

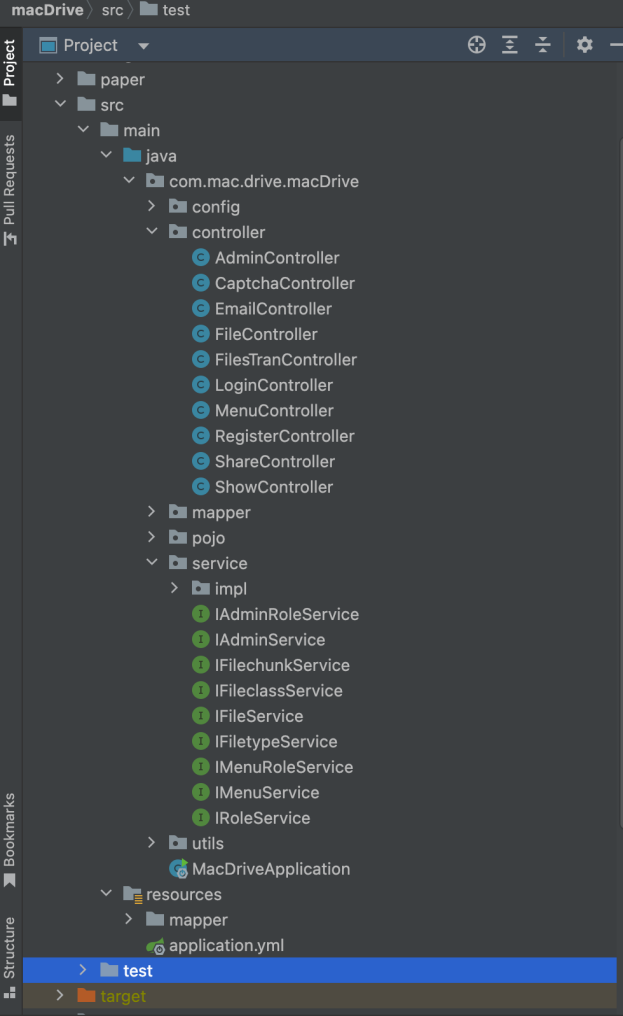


Figure5.2 macDrive project Layer

### 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.

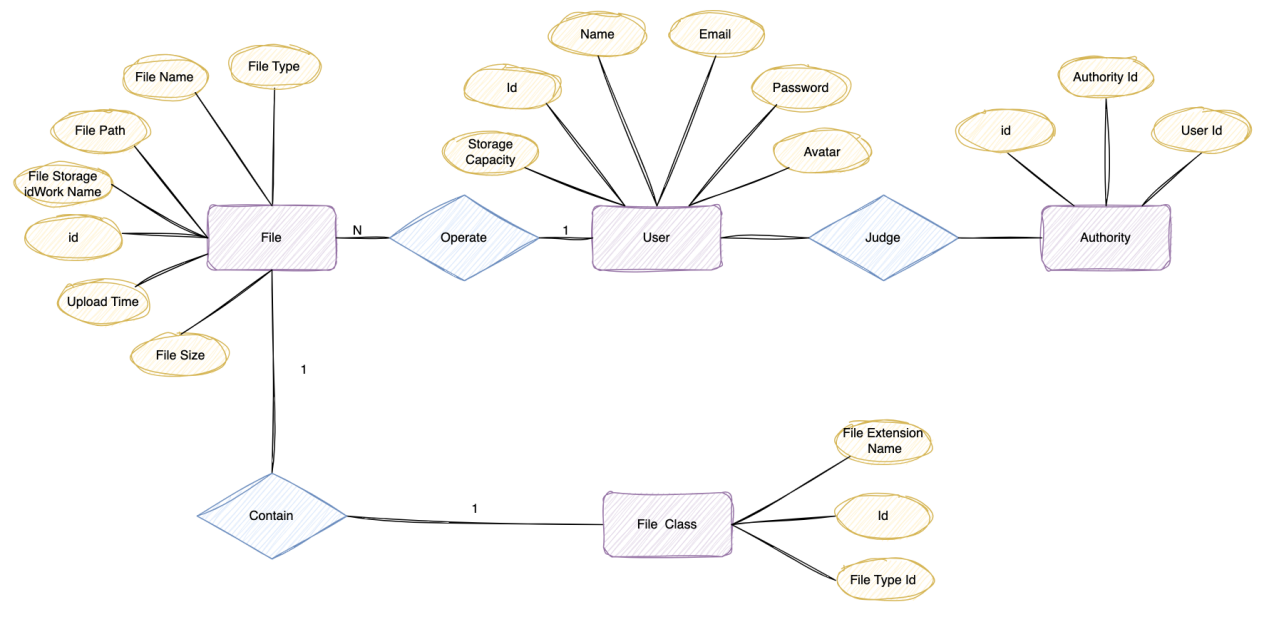


Figure 5.3 Overall ER Diagram

## 5.2 Functional Implement

## 5.2.1 User Domain

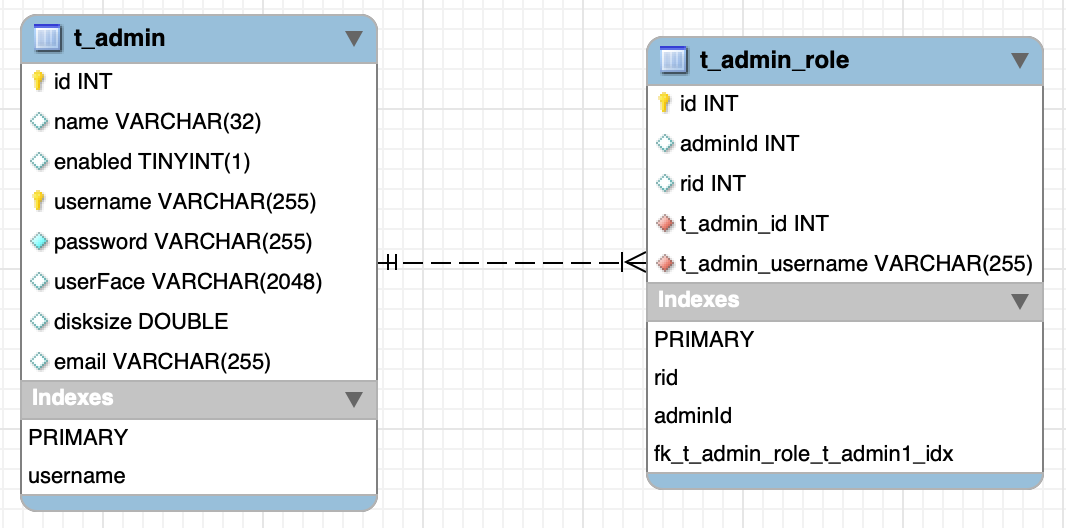


Figure 5.4 User ER Detail Diagram

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. First of all, the table gathers the following data on the user: his/her name, username, password, and other thing necessary for authentication. Secondly, it collects some unique data, e.g., avatar as well as 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.

While it is common for an admin role to have multiple users associated with it, the "t\_admin\_role" table determines different roles that will be assigned to those multiple users and the system. For application, each role is identified by a unique "rid" that relates to the user's "id". This joyful is binding for the system to provide defined responsibilities and privilege rules for different users. This mechanism for authorization of roles has foundational value as it comprises a mean of accurate control over graduation of administrator rights.

The two tables of users share the attributes "t\_admin\_id" and "t\_admin\_role\_adminId", which likewise show how system consistency and integrity are considered to be preserved through the use of these fields. In practical terms, when administrator information is up-to-date in "t\_admin", appropriate records in "t\_admin\_role" will automatically be referencing administrator information and role, respectively, ensuring data consistency. This meticulous and detailed database implementation will enable high-performance and secure websites' hosting service.

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

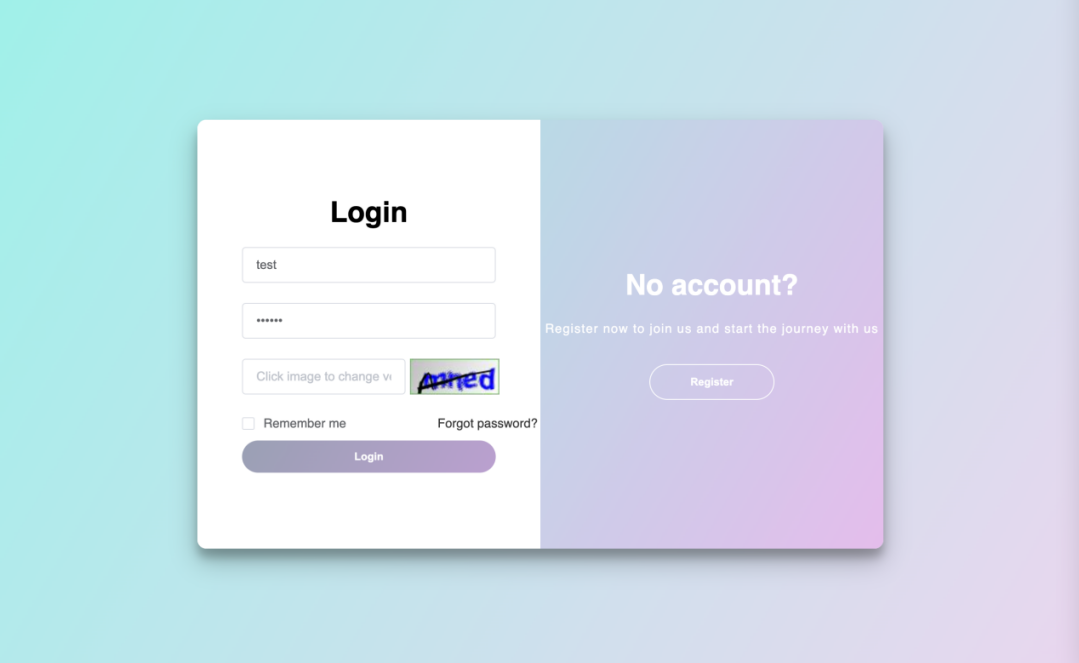


Figure 5.5 User Login Page

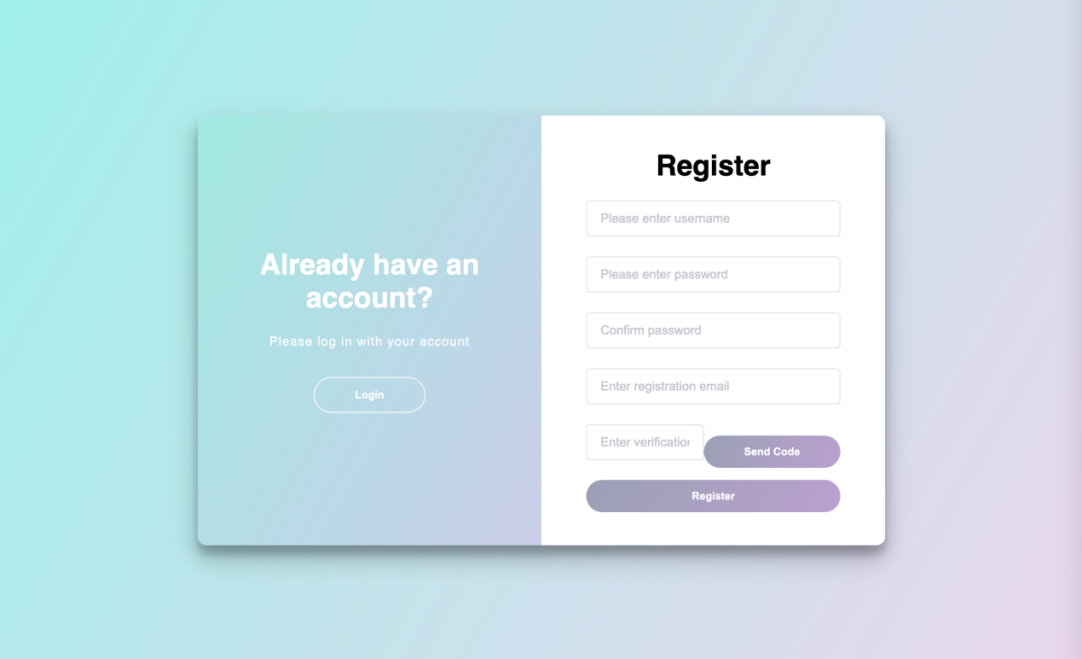


Figure 5.5 User Register Page

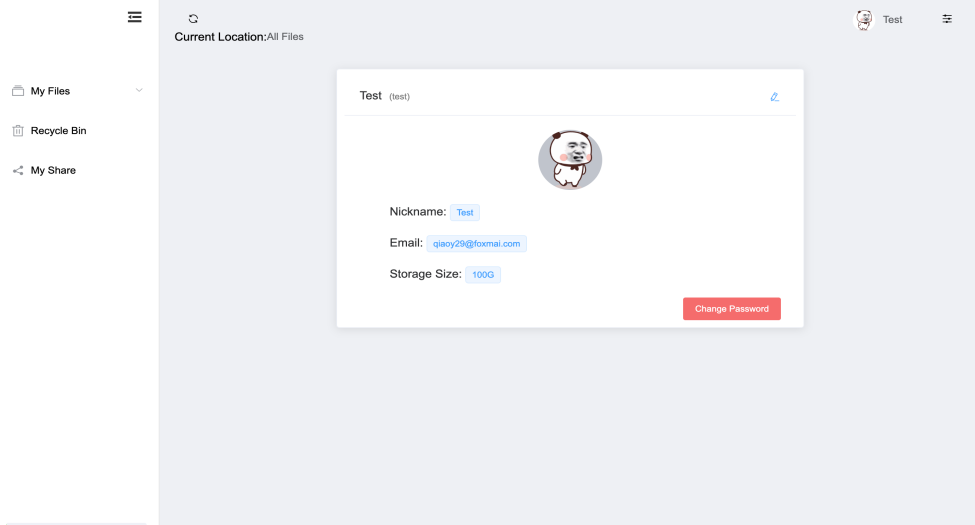


Figure 5.6 User Infomation Detail Page

## 5.2.2 File Domain

The t\_file is indicated in "t\_file" table, where the filenames, type, storage information, creating time, start time, and time deleted are listed. These detailed records not only ensure file tracking and management, but also facilitate file operations such as deletion and restoration. The attachment "t\_file" is also linked with the "t\_fileclass" and "t\_filetype" table through foreign keys, where "t\_fileclass" registers the class of films or the kind of movie for that matter.

"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.

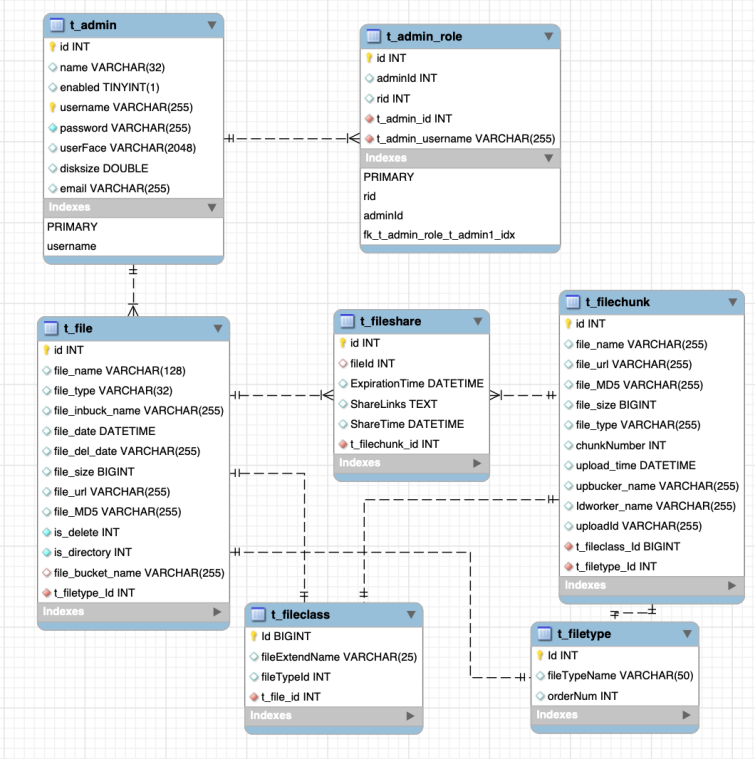


Figure 5.7 File ER Detail Diagram

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：

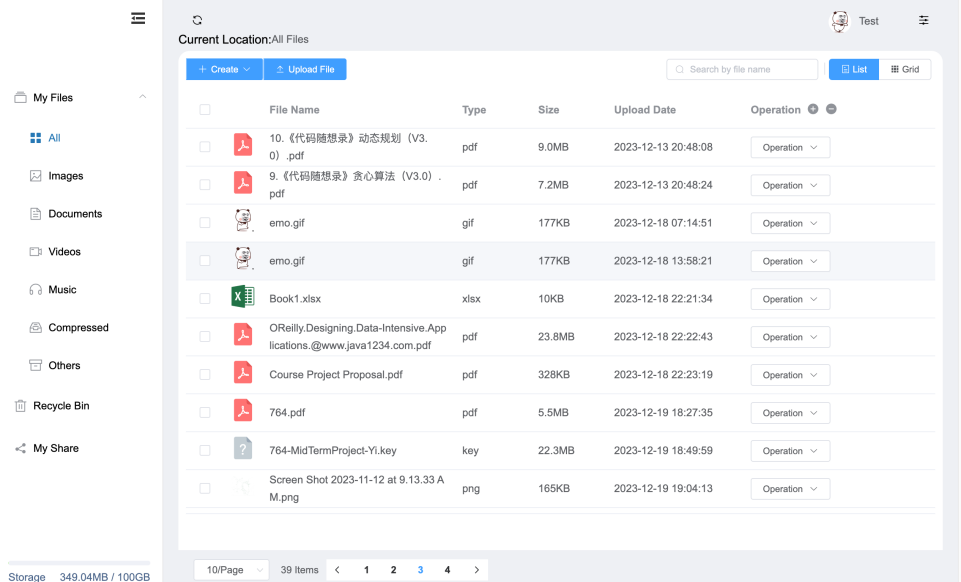


Figure 5.8 File List

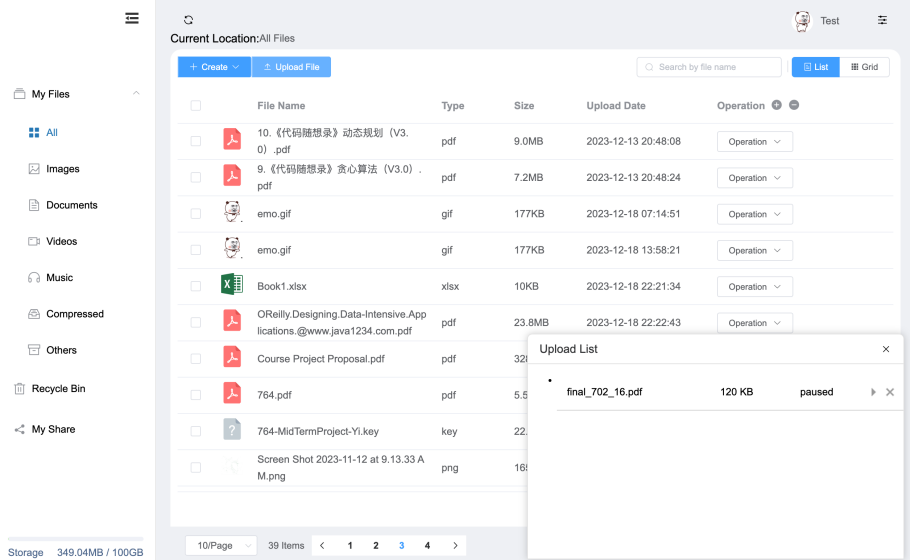


Figure 5.9 File Upload(in process status)

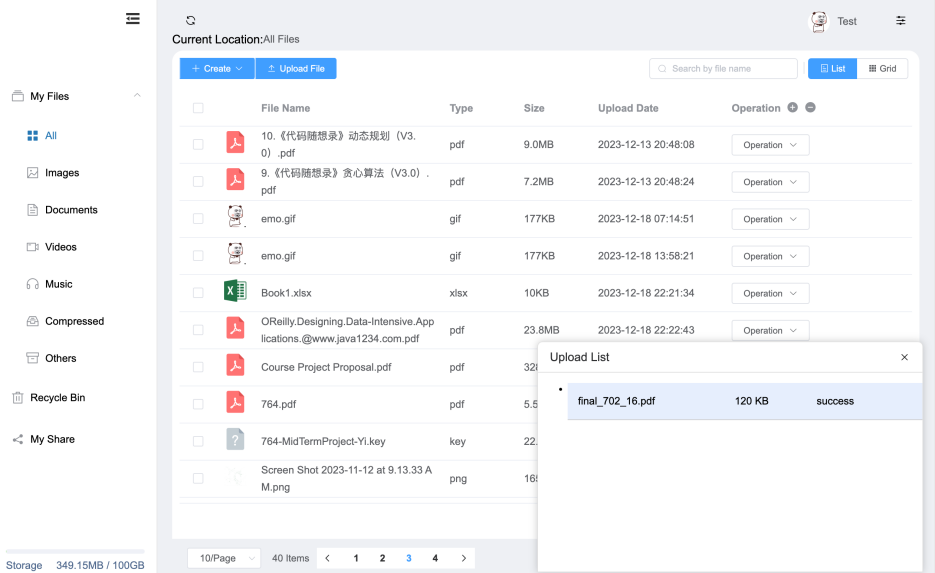


Figure 5.10 File Upload(done status)

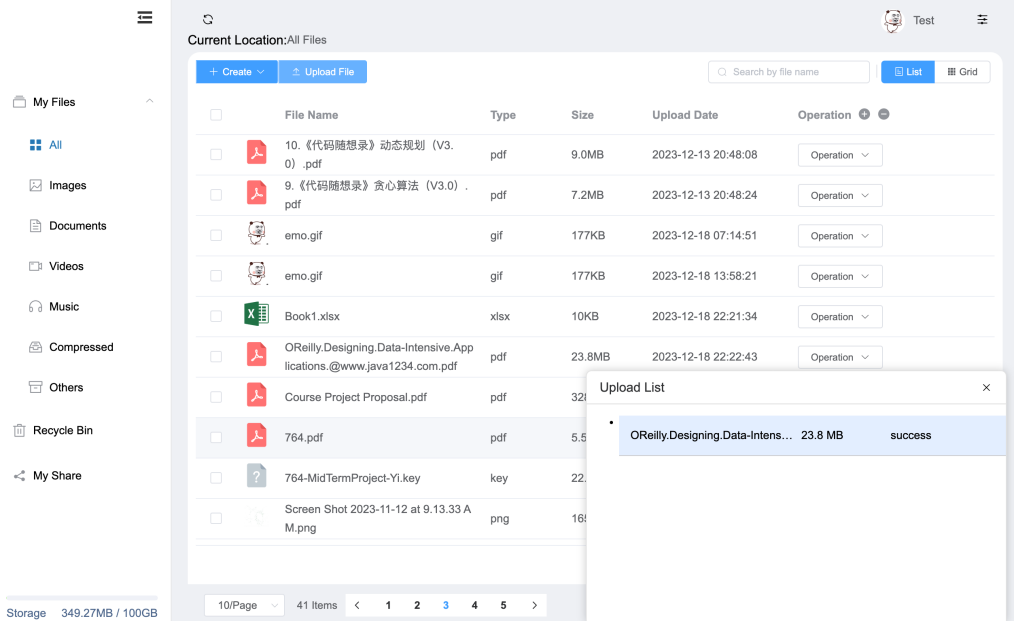


Figure 5.11Big File Chunk Upload

## 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.

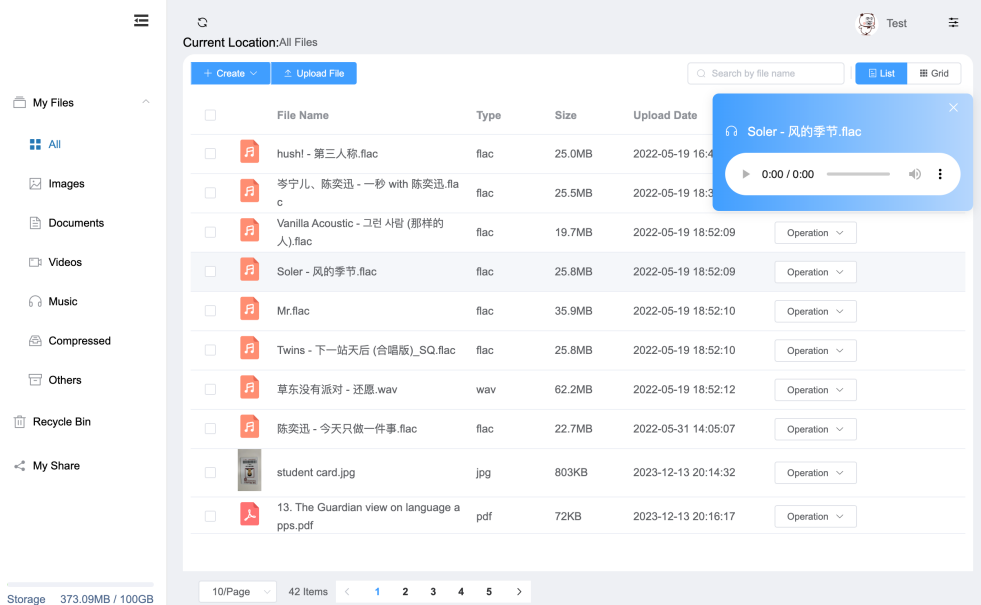
The following screenshot shows the actual implementation of the system, which are 5.12 Music Preview 5.13Picture Preview：

Figure 5.12 Music Preview

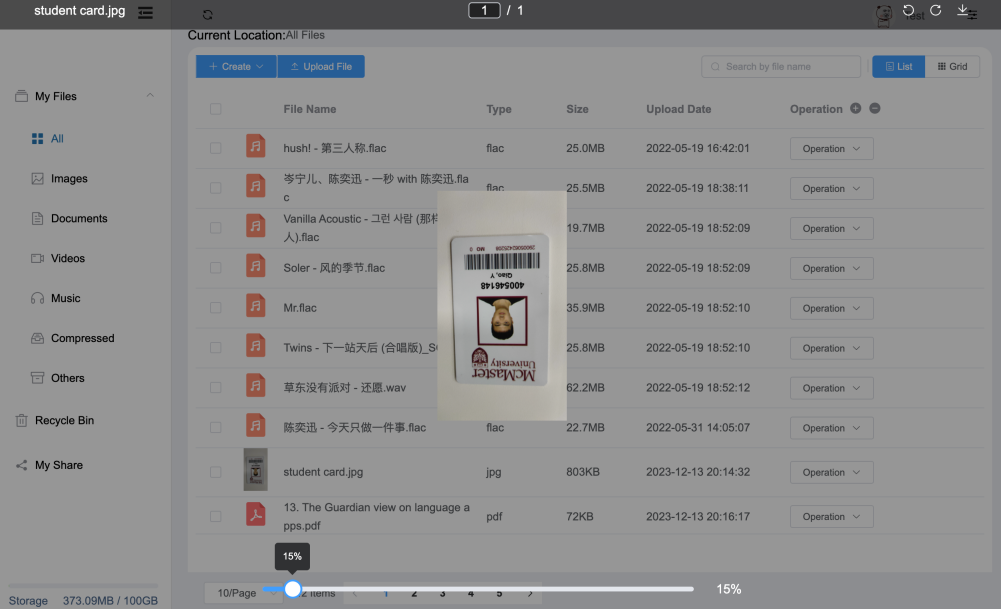


Figure 5.13 Picture Preview

# 5.2 Interface Development

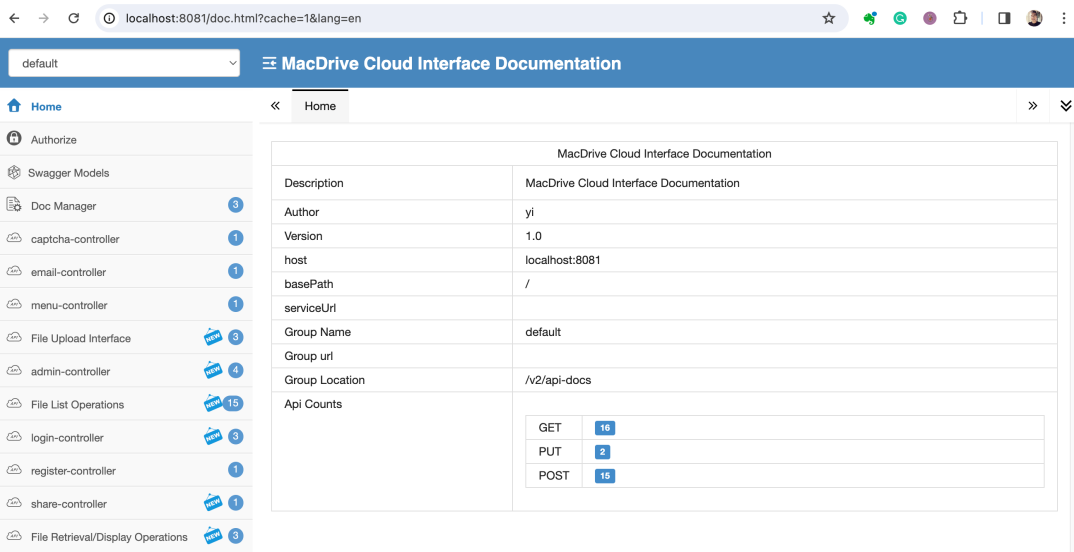


Figure 5.14 Back-end Interface Display with Swagger

In terms of code structure, I'll be using Swagger to demonstrate interface documentation. Swagger is an open source toolset that uses the OpenAPI specification to help developers document and test APIs.The OpenAPI specification is a standardized way of describing web-based APIs, including their functionality, parameters, return types, and more[[21]](#endnote-20)[21].The Swagger toolset includes several different tools such as Swagger UI, Swagger Codegen, and Swagger Editor. The Swagger toolset includes several different tools such as Swagger UI, Swagger Codegen, and Swagger Editor[[22]](#endnote-21)[22].

When input the URL: <http://localhost:8081/doc.html?cache=1&lang=en>, the Swagger UI will show. There are 22 file domain interfaces, 9 user domain interfaces, 3 display interfaces and 7 other interfaces.

# 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. [[23]](#endnote-22)However, writing high-quality unit tests is still not a trivial task[[24]](#endnote-23)[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 enhance the testability, we employ the DI design approach, which assists in replacing and simulating external dependencies and ensures a more accurate unit testing process. Additionally, we took advantage of Spring Boot's practical testing frameworks, having test annotations and tools in a rich set that helped us in the automation of testing to verify the interface behavior in a correct way.

As Vue.js front-end development is concerned, we achieve the module-wise structure for our code testing as one of the basic principles of our development. Every component is isolated and prone to be independently run to ensure that the output is right, so the other components are not relied upon. We will use tools like Vue Test Utils, which can simulate user interactions in different situations, to validate the component accuracy and functionality.

## 6.2 Writing and Structuring Testing

Testing is not only a phase in code development but an ever-present activity across the software lifetime. We set a test-driven development (TDD) approach, where the total coding process is completely dependent on the first creation of the test case and followed by implementation of the functionality in order to satisfy those tests. The approach also not only detects potential problems in time but also encourages one to design to do it right. In Spring Boot application testing, we use JUnit and Mockito frameworks to run unit tests. They ensure that the business rules in methods are right and that the dependencies are mocked.

As crucial parts of our application testing plan, the integration tests make certain whether the components work together as expected. We are making use of Spring Boot's test suite to focus on the integration tests which cover the whole flow from database operations into RESTful API requests. Back-end testing was done using Karma and Jest frameworks, which enable unit and end-to-end (E2E) testing to guarantee that the logic of the user interface is correct.

## 6.3 Unit Testing

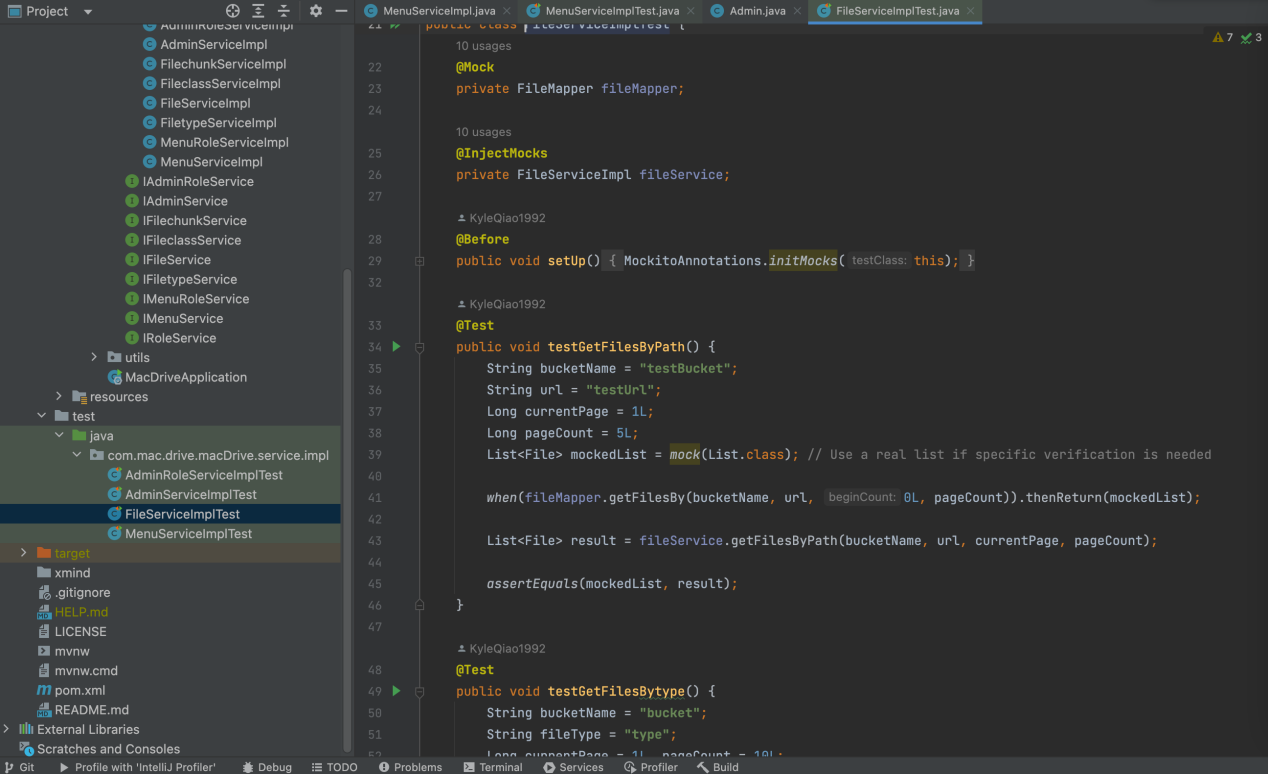


Figure 6.1 Unit Test

I have put a spotlight on unit testing here – this research is all about the web-based system that we have developed. Through JUnit, a combination of methods was employed to carry out a thorough unit testing of the back-end service, which was an assurance of the functional correctness and the stability of each module at the unit level. The unit tests we wrote at first will be following the principles behind test-driven development (TDD), where the business code is written after the test cases have been written. It serves to expose potential pitfalls of logic and integration errors in the early phase of development and also guides to a cleaner and well-maintained code.

For code quality and future scalability, a unit test concentrates on the functional parts, which should be centered on small areas of the code. The file upload function was tested by creating several test cases to see how the system will behave when given a different type of file, different data size, and under the two upload scenarios (Ethernet and wireless). For the user authentication function, we explored the different possibilities by performing the normal login, wrong password, and account lockout tests. Our purpose was to achieve the maximum possible test coverage so that all the coding was done right.

During the testing phase, we extensively relied on the Mockito framework to emulate the external dependencies, such as third-party services and real database connections, that helped us validate the business logic even without using them in the tests. This method not only expedites the run of test by computer but also guarantees the integrity of results. On the one hand, we do it through our Continuous Integration (CI). And it is also the case that the unit tests are automatically run after each code commit. This automated testing process helps catch regression errors in a timely manner and also greatly improves the efficiency of the development process and the stability of the code. All in all, unit testing, as the basis of quality assurance, was the most crucial element in developing our web hosting system.

## 6.4 End-to-End Testing

Through unit testing and E2E testing (end-to-end testing), a possible malfunctions of a web hosting system should be detected, and the whole process must go smoothly and meet the expectations of the user. Unit testing, which is the process of testing individual components for their correctness, is different from E2E testing, which tests the entire system from a user action perspective and verifies the behavior of the whole application.

Postman, as a part of end-to-end testing, is included to be an effective API testing tool. With Postman, we can submit different HTTP requests in order to mimic the interactions between the client and the netbook system. Postman is mainly doing these API endpoint responsiveness and accuracy verification in the web hosting system, like testing the file upload task through POST request that carries specific key-value pairs as well as file upload file, or retrieving the files through GET request. Postman is generous compared to the other plugins. It has varieties of features like you can set your own environment variables, write scripts of the tests, even simulate user authentication process.

On the other hand, Postman's main advantages include various Collections and Environment features. With these, we can create a set of grouped requests and switch between different test environments (or as we like to say, circuits), which leads to a more enhanced testing capability. Thus, we can only use one test collection for developing, testing, and production environments and simply switch between various environments, thus ensuring that we do not have testing issues and scalable testing. In our end-to-end testing practice, we used the automatic test run of Postman for collections, and the most assured way to validate the API response is the use of built-in library of test assertions.

The process of the E2E testing cannot be beyond the situation when the model is applied as much like as it is in the production environment where the environment includes databases, storage services, and third parties' integrations. It is being a pathway to detect the problems that might appear only in some environment configurations. We also conducted E2E testing of the entire web hosting system with the user in mind to determine its availability. The testing output from our end-to-end testing system will allow us to have a good measure of whether the web hosting system meets our quality targets or not.

## 6.5 Performance Testing

I used a flame diagram, an efficient visualization tool, to visualize the CPU resource consumption of the system at runtime. The flame diagram reveals the execution time share of each method in the system, which provides a basis for us to identify potential performance bottlenecks.

After an in-depth assessment of the frequency and response time of the infamous method calls, we found that the post-processing phases of bean creation and setup of attributes are pretty valuable places for a performance boost. If poorly tuned, the method calls during this period can cause longer application startup time and the slow in responsiveness of the system as a whole.

Besides informing us of critical parts and methods that lead to unnecessary resource consumption, this benchmark is also about specifying the performance thresholds. For example, methods such as "populateBean" and "instantiateBean" are particularly prominent during object instantiation, suggesting that we may need to optimize the construction and configuration of beans, such as using lazy loading strategies or adjusting the scope of the bean.

To summarize, the flame diagram provides us with a direct and efficient way to evaluate and analyze the performance of a web hosting system. Through in-depth interpretation of the flame diagram, we are able to prioritize optimization and take targeted measures, such as code refactoring and resource management strategy adjustment, in order to achieve the goal of improving system performance and optimizing user experience. In the future, we will continue to use the Flame Diagram as a regular performance evaluation tool to ensure that the Nethub system can maintain efficient and stable performance while continuously developing and expanding its functionality.

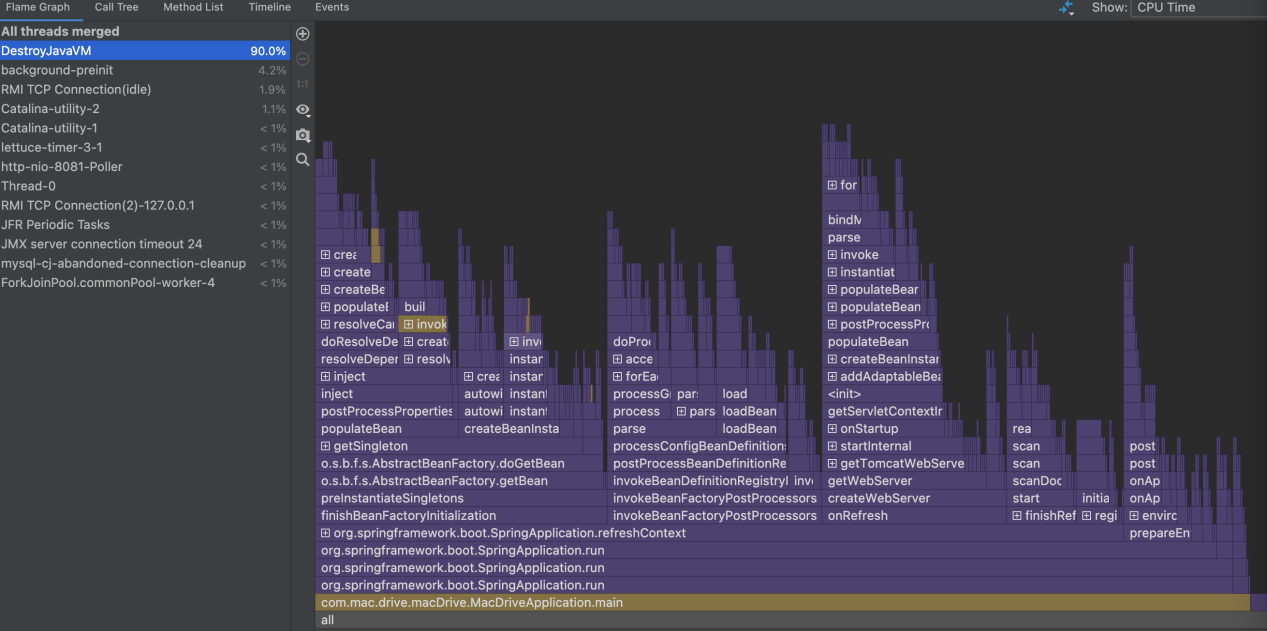


Figure 6.2 Flame Diagram of MacDrive

# 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.

    [21] Nienaber, C., & Suter, R. (2023, September 11). ASP.NET Core web API documentation with Swagger / OpenAPI. Microsoft Learn. Retrieved from https://learn.microsoft.com/en-us/aspnet/core/tutorials/web-api-help-pages-using-swagger?view=aspnetcore-6.0

    [22] Swagger. (n.d.). Swagger UI. Retrieved from https://swagger.io/tools/swagger-ui/ [↑](#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)
23. [↑](#endnote-ref-22)
24. [↑](#endnote-ref-23)