
UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI



BACHELOR THESIS

Development of Data Lake APIs and Dashboard

Author:

Nguyen Phuong Thao
USTHBI9-212

Supervisor:

Dr. TRAN Giang Son, USTH ICT Lab

Hanoi - July 2021

A blue ink signature, likely of the author, Nguyen Phuong Thao. It is a stylized, cursive signature.

Agreed to submit for defense

A red ink signature, likely of the supervisor, Dr. TRAN Giang Son. It is a stylized, cursive signature.

Trần Giang Sơn

Acknowledgments

Throughout the writing of this dissertation, I have received a great deal of support and assistance.

I would like to thank my supervisor, Professor Tran Giang Son, whose expertise was invaluable in formulating my methodology. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level.

I would like to acknowledge my professors from my university for their excellent teaching. I want to thank you for your patient support and for all of the opportunities I was given.

In addition, I would like to thank my parents. Finally, I could not have completed this dissertation without the support of my friends, who provided stimulating discussions as well as happy distractions to rest my mind outside of my work.

Contents

Acknowledgments	i
Contents	ii
List of Tables	iv
1 Introduction	1
1.1 Context and Motivation	1
1.2 Thesis Structure	2
2 Objectives	4
2.1 Desired Features	4
2.2 Expected Outcome	6
3 Requirement Analysis	7
3.1 Overall System Requirements	7
3.2 Users & Non-functional Requirements	7
3.3 Use Cases	8
3.3.1 Use Cases Diagram	8
3.3.2 Users Characteristics	10
3.4 Use Case and Scenario Description	10
3.4.1 Use case: Register (SUB-FEATURE 1.1)	10
3.4.2 Use case: Login (SUB-FEATURE 1.2)	12
3.4.3 Use case: Manage Users (FEATURE 2)	13
3.4.4 Use case: Manage User Groups (FEATURE 3)	16
3.4.5 Use case: Manage Folders (FEATURE 4)	19
3.4.6 Use case: Manage Files (FEATURE 5)	23
3.4.7 Use case: Manage ACLs (FEATURE 6)	27
4 Methodology	30
4.1 System Architecture	30
4.2 Database Design	33
4.3 Use Cases Implementation	42

4.3.1	Login	42
4.3.2	Register	43
4.3.3	Manage Users	44
4.3.4	Manage User Groups	47
4.3.5	Manage Folders	50
4.3.6	Manage Files	55
4.3.7	Manage ACLs	63
4.4	Tools and Techniques	66
4.4.1	SQL	66
4.4.2	Spring Boot	66
4.4.3	ReactJS	66
4.4.4	Axios	67
4.4.5	Material-UI	67
4.4.6	Swagger	67
4.4.7	JWT (JSON Web Token)	67
4.4.8	RBAC and ACL Authorization	68
5	Results and Discussion	69
5.1	Result	69
5.1.1	API	69
5.1.2	Web Dashboard	71
5.2	Discussion	74
6	Conclusions	75
6.1	Conclusion	75
6.2	Future Works	75
	Glossary	76
	Acronyms	77
	Bibliography	78
	Appendices	79

List of Tables

3.1	Register Basic Flow (SUB-FEATURE 1.1)	11
3.2	Login Basic Flow (SUB-FEATURE 1.2)	12
3.3	Manage Users Basic Flow (SUB-FEATURE 2.1 & SUB-FEATURE 2.2)	14
3.4	Update User Sub Flow (SUB-FEATURE 2.3)	15
3.5	Delete User Sub Flow (SUB-FEATURE 2.4)	15
3.6	Manage Groups Basic Flow (SUB-FEATURE 3.1 & SUB-FEATURE 3.1)	17
3.7	Update Group Sub Flow (SUB-FEATURE 3.3)	18
3.8	Delete Group Sub Flow (SUB-FEATURE 3.4)	18
3.9	Manage Folders Basic Flow (SUB-FEATURE 4.1 & SUB-FEATURE 4.2)	20
3.10	Edit Folder Sub Flow (SUB-FEATURE 4.3)	21
3.11	Move a sub-folder to a Folder Sub Flow (SUB-FEATURE 4.4)	21
3.12	Delete Folder Sub Flow (SUB-FEATURE 4.5)	22
3.13	Manage Files Basic Flow (SUB-FEATURE 5.1 & SUB-FEATURE 5.2)	24
3.14	Update File Sub Flow (SUB-FEATURE 5.3)	25
3.15	Add a File to a Folder Sub Flow (SUB-FEATURE 5.4)	25
3.16	Download File Sub Flow (SUB-FEATURE 5.5)	26
3.17	Delete File Sub Flow (SUB-FEATURE 5.6)	26
3.18	Manage ACLs Basic Flow (SUB-FEATURE 6.1 & SUB-FEATURE 6.2)	28
3.19	Delete ACL Sub Flow (SUB-FEATURE 6.3)	29
4.1	Database User Design	35
4.2	Database Role Design	35
4.3	Database User Role Design	36
4.4	Database Group Design	36
4.5	Database User Group Design	37
4.6	Database Refresh Token Design	37
4.7	Database Folder Design	38
4.8	Database File Design	39
4.9	Database ACL Class Design	39
4.10	Database ACL Sid Design	40
4.11	Database ACL Object Identity Design	40
4.12	Database ACL Entry	41
4.13	Database ACL Design	42

5.1 All API Endpoints	69
---------------------------------	----

List of Figures

1.1	Level 1: System Context diagram: ComLake Overall Architecture diagram	2
3.1	Use cases diagram	9
4.1	Container diagram for ComLake System - Backend API and Dashboard . .	31
4.2	Component diagram for ComLake Sytem - Backend API	32
4.3	API Backend Application Package Class diagram	33
4.4	Overall Database Design	34
4.5	Login Sequence diagram	43
4.6	Register Sequence diagram	44
4.7	Manage Users - Basic Flow Sequence diagram	45
4.8	Manage Users - Update User Sub Flow Sequence diagram	46
4.9	Manage Users - Delete User Sub Flow Sequence diagram	47
4.10	Manage Groups - Basic Flow Sequence diagram	48
4.11	Manage User Groups - Update Group Sub Flow Sequence diagram	49
4.12	Manage User Groups - Delete Group Sub Flow Sequence diagram	50
4.13	Manage Folders - Basic Flow Sequence diagram	51
4.14	Manage Folders - Update Folder Sub Flow Sequence diagram	52
4.15	Manage Folders - Delete Folder Sub Flow Sequence diagram	54
4.16	Manage Folders - Move a Sub-folder to a Folder Sub Flow Sequence diagram	55
4.17	Manage Files - Basic Flow Sequence diagram	57
4.18	Manage Files - Update File Sub Flow Sequence diagram	58
4.19	Manage Files - Delete File Sub Flow Sequence diagram	59
4.20	Manage Files - Move a File to a Folder Sub Flow Sequence diagram	61
4.21	Manage Files - Download File Sub Flow Sequence diagram	62
4.22	Manage ACLs - Basic Flow Sequence diagram	63
4.23	Manage ACLs - Delete ACL Sub Flow Sequence diagram	65
5.1	List all files and folders user interface	72
5.2	Upload File user interface	72
5.3	View File Details user interface	73
5.4	Edit File user interface	74

Chapter 1

Introduction

1.1 Context and Motivation

Data-driven decision-making is changing how we live and work. For every aspect of data science, machine learning, and advanced analytics, data scientists require data to help make decisions [1]. Big Tech companies like Google and Facebook to medium financial services organizations and insurance companies have always been data-driven. Big Data and Data Science are permeating all aspects of our lives.

The concept of a Data Lake was driven by particular challenges that organizations were facing with the way data was handled, processed, and stored. Initially, what we used to is a data puddle, which is a single-purpose or single-project, started maintaining vast amounts of data themselves with almost no reuse in other applications in the same organization—these created information silos across various platforms. A demand was felt to have Data Lakes, which could solve data governance, data ownership, and data accessibility. A data lake could store any form of data, data as-is, data have no fixed schema,... so it could be analyzed and kept ready for consumption by consumer applications.

Many researchers at the University of Science and Technology of Hanoi (USTH) and the Vietnam Academy of Science and Technology (VAST) work with data regularly. Suppose a dataset is frequently researched by multiple researchers from various departments and periods. However, data is currently arranged manually, even on the laboratory's storage, leading to duplication and difficult data discovery.

In this project, we attempt to develop a Data Lake to solve those challenges.

To start, we will use the C4 model to describe the software architecture because this application belongs to a more extensive system of ComLake, developed by a group of the software development team, supervised by Professor Tran Giang Son.

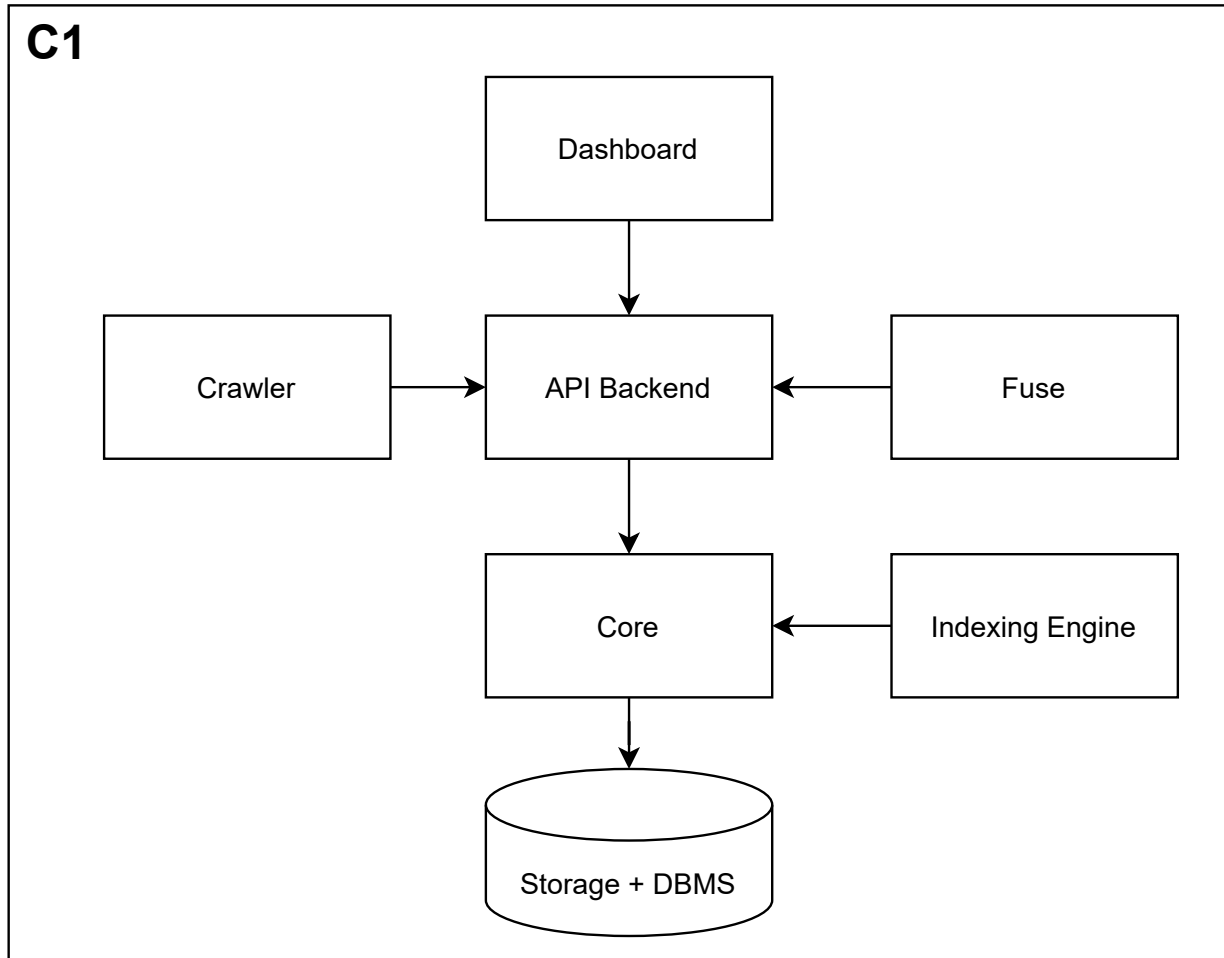


Figure 1.1: Level 1: System Context diagram: ComLake Overall Architecture diagram

This is the System Context diagram for the ComLake system. The system comprises seven containers: a web application Dashboard, a server-side API Application, a data web Crawler, a Filesystem in USERspace (FUSE), an Image Indexing Engine using Machine Learning, a Core service that encapsulates distributed file system (DFS), and a database management system (DBMS).

The Dashboard, a ReactJS application that runs in the customer's web browser, will be the end-first user's point of contact. The Public API (API Backend in the figure) would handle authentication and authorization for external users and most internal services before being modified and given to the core API.

We will strive to build the Public API and Dashboard container.

1.2 Thesis Structure

The thesis is divided into six main chapters to explain the development of Data Lake APIs and Dashboard.

- The first chapter, Introduction, will discuss the fascinating subject of Data Lake APIs and Dashboards and the proposed solution in Chapter 1: Introduction.
- The second chapter, Objectives, we shall establish the project's standard requirements, present a quick summary of the system's operation, and present the reasons for its development.
- The third chapter, Requirement Analysis, will continue with the system's scenarios, use cases, object models, and dynamic models.
- The fourth chapter, Methodology, includes the entire functional specification and navigational paths that reflect the screen sequence. We will go through all of the tools and techniques utilized in the project, why they were chosen, and how the use cases were implemented in detail.
- The fifth chapter, Result and Discussion, we shall detail all of the functionalities that are implemented in the system
- The sixth chapter, Conclusions, provides conclusions about the project outcome and discusses the difficulties that exist during project development.

Chapter 2

Objectives

In this segment, we will define the standard requirements of the project, provide a brief overview of the function of the system and the reasons for its development.

2.1 Desired Features

The main goal of this project is to develop a Data Lake APIs and dashboard with basic services:

- **FEATURE 1: Authenticate**
 - **SUB-FEATURE 1.1: Register:** allow users to create a new account
 - **SUB-FEATURE 1.2: Login:** allow users to access into the system.
- **FEATURE 2: Manage Users**
 - **SUB-FEATURE 2.1: Create User:** allow Admin to create an account
 - **SUB-FEATURE 2.2: View User:**
 - * Allow Admin to get a list of all users in the system, filter users by their metadata, export users data to a CSV file
 - * Allow (Regular) User to view their own profile
 - **SUB-FEATURE 2.3: Update User:**
 - * Allow Admin to edit users' information
 - * (Regular) User to update their own profile
 - **SUB-FEATURE 2.4: Delete User:** allow Admin to delete an account
- **FEATURE 3: Manage User Groups**
 - **SUB-FEATURE 3.1: Create Group:** allow Admin to create a group

- **SUB-FEATURE 3.2: View Group:** allow Admin to get a list of all groups in the system, filter groups by their metadata, export groups data to a CSV file
- **SUB-FEATURE 3.3: Update Group:** allow Admin to add a user to a group
- **SUB-FEATURE 3.4: Delete Group:** allow Admin to delete a group
- **FEATURE 4: Manage Folders**
 - **SUB-FEATURE 4.1: Create Folder:** allow users to create a folder
 - **SUB-FEATURE 4.2: View Folder:**
 - * Admin can view all folders
 - * A (regular) User could only access folders they have permission to read
 - * Users can filter folders by their metadata, and export folders data to a CSV file
 - **SUB-FEATURE 4.3: Edit Folder:**
 - * Admin can edit all folders
 - * A (regular) User could only edit folders they have permission to write
 - **SUB-FEATURE 4.4: Add a sub-folder to a folder:**
 - * Admin can add all sub-folders to all folders
 - * A (regular) User could only add a sub-folder to a folder they have permission to write
 - **SUB-FEATURE 4.5: Delete Folder:**
 - * Admin can delete all folders
 - * A (regular) User could only delete folders they have permission to write
- **FEATURE 5: Manage Files**
 - **SUB-FEATURE 5.1: Create File:** allow users to create a file
 - **SUB-FEATURE 5.2: View File:**
 - * Admin can view all files
 - * A (regular) User could only access files they have permission to read
 - * Users can filter files by their metadata, and export folders data to a CSV file
 - **SUB-FEATURE 5.3: Update File:**
 - * Admin can edit all files
 - * A (regular) User could only edit files they have permission to write
 - **SUB-FEATURE 5.4: Add a file to a folder:**
 - * Admin can add all files to all folders
 - * A (regular) User could only add files to folders they have permission to write

- **SUB-FEATURE 5.5: Download File:**
 - * Admin can add download all files
 - * A (regular) User could only download files they have permission to read
- **SUB-FEATURE 5.6: Delete File:**
 - * Admin can delete all files
 - * A (regular) User could only delete files they have permission to write
- **FEATURE 6: Manage ACLs**
 - **SUB-FEATURE 6.1: Grant ACL:** allow users to grant ACL (permission to read or write) to a user or a group for a file or folder
 - **SUB-FEATURE 6.2: View ACL:** allow users to view all ACL (permission to read or write) of all folders or files they have access to; users could also search in ACL sources, filter ACLs based on their metadata, and export ACL data to a CSV file
 - **SUB-FEATURE 6.3: Delete ACL:** allow users to remove ACL (permission to read or write) of a user or a group in a file or folder

2.2 Expected Outcome

The system aims to create a Data Lake for all uses and storage of USTH and beyond the university. The specific goals include:

- Develop a backend application that has all the desired features discussed and connected to the Data Lake Core
- Develop a web interface with interactions between users and the system
- The web page should be able to run in multiple browsers such as Chrome, Firefox, Safari

Chapter 3

Requirement Analysis

In this chapter, we will examine a brief review of the project's functions and the system's scenarios and use cases. This section includes the entire functional specification.

3.1 Overall System Requirements

In general, this application should satisfy the following requirements:

- A login system with authentication
- Users can upload and download files
- The system limited users with some features
 - For Admin:
 - * Manage Users
 - * Manage User Groups
 - * Access to all files and folders
 - * Grant or remove permissions for all files and folders
 - For Regular Users:
 - * View their own profile
 - * Update their own profile
 - * Access to files and folders they have permission to read and/or write
 - * Grant or remove permissions for the files or folders they own

3.2 Users & Non-functional Requirements

- **Users Requirement:** Have a device with internet access, preferable Chrome web browser

- **Performance:** The website is accessible by anyone with a public domain.
- **Availability:** The user can only work with the system fully if they can login with a registered account.
- **Re-usability:** User can use external files with the website. User can use CSV file generated from the website for other purposes. The data generated by the website can be use as the resource for other services.
- **Reliability:** The system will not work without the Internet connection.

3.3 Use Cases

3.3.1 Use Cases Diagram

The following diagram describes the functions that a user can perform with the system.

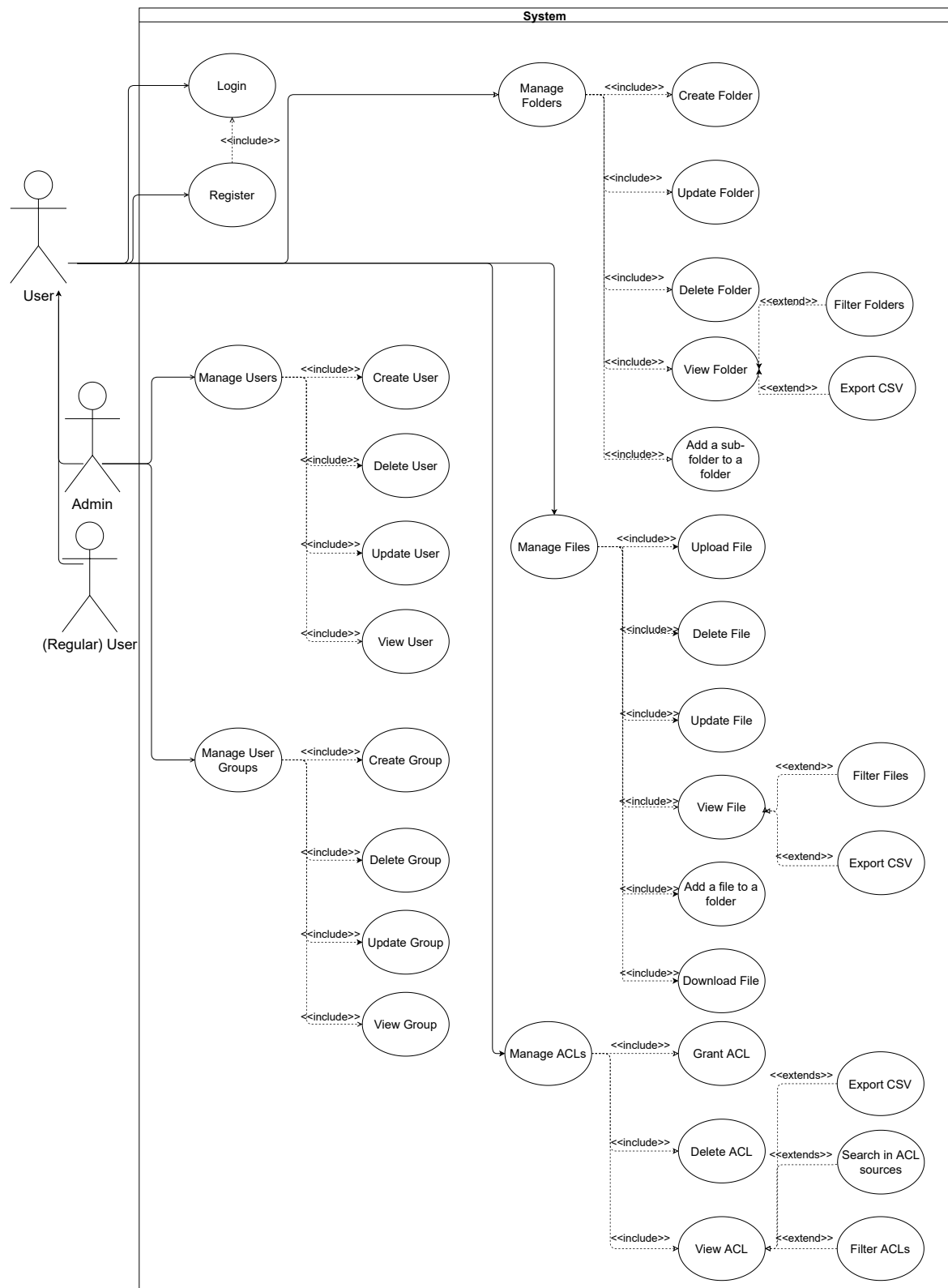


Figure 3.1: Use cases diagram

3.3.2 Users Characteristics

Two types of users interact with the system: **Admin** and (regular) **User**. Each user type has different uses in the system, so each of them has its requirements:

- **Admin**: a user who has access to all resources; only admin could manage user groups, total management of users
- **User**: a user who only has access to the resource they have permission to use, only could view and edit their profile.

3.4 Use Case and Scenario Description

3.4.1 Use case: Register (SUB-FEATURE 1.1)

Brief description

This use case describes how a user registers to the system.

Flow of events

1. Basic Flow

The table 3.1 takes place when one person wants to create a new account on the system.

Actor Action	System Action	Data
	1. The system requests that the Actor enter required information	- First name - Last name - Email - Username - Password
2. The Actor enters their information.	3. The system check the entered account exists on the system or not with username and email	
	4. The system validates user's information and create a new account for user and log the Actor into the system after created successfully	

Table 3.1: Register Basic Flow (SUB-FEATURE 1.1)

2. Alternative Flows

(i) Account exists

If in the Basic Flow, at step 3, the account has existed on the system, the system displays an error message. If the user goes back, return the form. The Actor can select to cancel the operation, at which point the Use Case ends.

(ii) Invalid Format or Insufficient Information

If, in the Basic Flow, at step 2, the Actor has not specified sufficient information to create a new account as required, the system will prompt the Actor for the invalid field or the missing information. The Actor can either enter the missing information or cancel the operation, at which point the Use Case ends.

Special Requirement

Only Regular Users can Register into the system. Admin is either pre-created or assigned an account.

Pre-conditions

User is not logged in.

Post-conditions

If the use case is successful, the user registers successfully. If not, the system state is unchanged.

3.4.2 Use case: Login (SUB-FEATURE 1.2)**Brief description**

The Use Case describes how a user logs into the system.

Flow of events

1. Basic Flow The table 3.2 describe when the Actor wishes to Login to the system.

Actor Action	System Action	Data
	1. The system requests that the Actor enter their username and password	- Username* - Password* * Required fields
2. The Actor enters their email and password.	3. The system validates the entered name and password and logs the Actor into the system.	

Table 3.2: Login Basic Flow (SUB-FEATURE 1.2)

2. Alternative Flows

(i) Invalid Username/Password

If in the Basic Flow, at step 2, the Actor enters an invalid username or password, the system displays an error message. The Actor can either return to the beginning of the Basic Flow or cancel the login, at which point the Use Case ends.

Special Requirement

None.

Pre-conditions

The user account must exist.

Post-conditions

If the Use Case was successful, the Actor is now logged into the system. If not, the system state is unchanged.

3.4.3 Use case: Manage Users (FEATURE 2)

Brief description

The Use Case allows the system Admin to manage users' information. This includes adding a new user to the system, changing existing users' information on the system, removing a user from the system, and viewing a user.

Flow of events

1. Basic Flow The table 3.3 describe when the Admin wishes to add, change, and/or delete User information in the system.

Actor Action	System Action	Data
1. The Admin wants to manage users.	2. The system shows list of all users.	List of all users details.
3. The Admin views the users lists and/or filters them by custom criteria.	4. The system displays the users by custom criteria.	List of all users with custom criteria.
	5. The system requests that the Admin specifies the function he/she would like to perform (either Create a User, Update User or Delete User).	- Manage selection
6. The Admin selects "Create User".	7. The system requests the Administrator to enter the User information.	- Username* - Password* - Email* - First Name - Last Name - Department - Affiliation * Required fields
8. The Admin provides the requested information.	9. The system generates and assigns a unique User Id number to the user. The User is added to the system.	

Table 3.3: Manage Users Basic Flow (SUB-FEATURE 2.1 & SUB-FEATURE 2.2)

*Update User Sub Flow*At step 6 of **Basic Flow**

Actor Action	System Action	Data
1. The Admin selects "Update User".	2. The system requests the Admin to select User from User list or User profile interface.	
3. The Admin selects User.	4. The system retrieves and displays the User.	
5. The Admin enter the desired changes to the User information.	6. The system updates the User record with the updated information.	- First Name - Last Name - Department - Affiliation

Table 3.4: Update User Sub Flow (SUB-FEATURE 2.3)

*Delete User Sub Flow*At step 6 of **Basic Flow**

Actor Action	System Action	Data
1. The Admin selects "Delete User".	2. The system requests the Admin to select User from User list or User profile interface.	
3. The Admin selects User.	4. The system retrieves and displays the User.	
5. The Admin select delete.	6. The system removes the User from the system.	

Table 3.5: Delete User Sub Flow (SUB-FEATURE 2.4)

2. Alternative Flows

(i) User not found

If in the Update User sub flow, at step 3 or Delete User sub flow, at step 3, a User with the specified id number does not exist, the system displays an error message. The Administrator can cancel the operation, at which point the Use Case ends.

Special Requirement

None.

Pre-conditions

The Admin must be logged into the system before this Use Case begins.

Post-conditions

If the Use Case was successful, the User information is created, updated, deleted from the system. Otherwise, the system state is unchanged.

3.4.4 Use case: Manage User Groups (FEATURE 3)

Brief description

The Use Case allows the system Admin to manage user groups' information. This includes adding a new group to the system, changing existing group' information on the system, removing a group from the system, and viewing a group.

Flow of events

1. Basic Flow The table 3.6 describe when the Admin wishes to add, change, and/or delete Group information in the system.

Actor Action	System Action	Data
1. The Admin wants to manage groups.	2. The system shows list of all groups.	List of all groups details.
3. The Admin views the groups lists and/or filter groups by custom criteria.	4. The system displays the groups by custom criteria.	List of all groups with custom criteria.
	5. The system requests that the Admin specify the function he/she would like to perform (either Create a Group, Update Group or Delete Group).	Manage selection
6. The Admin selects "Create Group".	7. The system requests the Administrator to enter the User information.	Name*
8. The Admin provides the requested information.	9. The system generates and assigns a unique Group Id number to the user. The Group is added to the system.	

Table 3.6: Manage Groups Basic Flow (SUB-FEATURE 3.1 & SUB-FEATURE 3.1)

*Update Group Sub Flow*At step 6 of **Basic Flow**

Actor Action	System Action	Data
1. The Admin selects "Update Group".	2. The system requests the Admin to select Group from Group list or Group details interface.	
3. The Admin selects Group.	4. The system retrieves and displays the Group along with a list of all users.	
5. The Admin adds a user from User list to a group.	6. The system updates the Group record with the updated information.	User-name

Table 3.7: Update Group Sub Flow (SUB-FEATURE 3.3)

*Delete Group Sub Flow*At step 6 of **Basic Flow**

Actor Action	System Action	Data
1. The Admin selects "Delete Group".	2. The system requests the Admin to select Group from Group list or Group details interface.	
3. The Admin selects Group.	4. The system retrieves and displays the Group.	
5. The Admin select delete.	6. The system removes the Group from the system.	

Table 3.8: Delete Group Sub Flow (SUB-FEATURE 3.4)

2. Alternative Flows

(i) Group not found

If in the Update Group sub flow, at step 3 or Delete Group sub flow, at step 3, a Group with the specified id number does not exist, the system displays an error message. The Administrator can cancel the operation, at which point the Use Case ends.

Special Requirement

None.

Pre-conditions

The Admin must be logged into the system before this Use Case begins.

Post-conditions

If the Use Case was successful, the Group information is created, updated, deleted from the system. Otherwise, the system state is unchanged.

3.4.5 Use case: Manage Folders (FEATURE 4)

Brief description

The Use Case allows the system users to manage folders' information. This includes adding a new folder to the system, changing existing folders' information on the system, removing a folder from the system, and viewing a folder.

Flow of events

1. Basic Flow The table 3.9 describe when the users wish to add, change, and/or delete Folder information in the system.

Actor Action	System Action	Data
1. The users want to manage folders.	2. The system shows the list of all folders and files details.	
3. The users view the folders and files lists and/or filter them by custom criteria.	4. The system displays the files and folders by custom criteria.	List of all files and folders with custom criteria.
	5. The system requests that the users specify the function he/she would like to perform (either Create a Folder, Update Folder, Move Folder, or Delete Folder).	- Manage selection
6. The users select "Create Folder".	7. The system requests the users to enter the Folder information.	- Name* - Source* - Topics* * Required fields
8. The users provide the requested information.	9. The system generates and assigns a unique Folder Id number to the User. The Folder is added to the system. The User is recognized as the Folder's owner by the system with read and write permission. Permission for Admin is also automatically added.	

Table 3.9: Manage Folders Basic Flow (SUB-FEATURE 4.1 & SUB-FEATURE 4.2)

*Edit Folder Sub Flow*At step 6 of **Basic Flow**

Actor Action	System Action	Data
1. The users select "Update Folder".	2. The system requests the users to select Folder from Folder list or Folder details interface.	
3. The users select Folder.	4. The system retrieves and displays the Folder.	
5. The users enter the desired changes to the Folder information.	6. The system updates the Folder record with the updated information.	- Name - Source - Topics

Table 3.10: Edit Folder Sub Flow (SUB-FEATURE 4.3)

*Move a Sub-folder to a Folder Sub Flow*At step 6 of **Basic Flow**

Actor Action	System Action	Data
1. The users select "Move Folder".	2. The system requests the users to select Folder from Folder list or Folder details interface.	
3. The users select Folder.	4. The system retrieves and displays the Folder.	
5. The users enter the destination folder name.	6. The system updates the Folder record with the updated information.	Destina- tion Folder name

Table 3.11: Move a sub-folder to a Folder Sub Flow (SUB-FEATURE 4.4)

*Delete Folder Sub Flow*At step 6 of **Basic Flow**

Actor Action	System Action	Data
1. The users selects "Delete Folder".	2. The system requests the users to select Folder from Folder list or Folder details interface.	
3. The users selects Folder.	4. The system retrieves and displays the Folder.	
5. The users select delete.	6. The system removes the Folder from the system.	

Table 3.12: Delete Folder Sub Flow (SUB-FEATURE 4.5)

2. Alternative Flows

(i) Folder not found

If in the Update Folder sub flow, at step 3 or Delete Folder sub flow, at step 3, a Folder with the specified id number does not exist, the system displays an error message. The users can cancel the operation, at which point the Use Case ends.

(ii) Invalid Format or Insufficient Information

If, in the Create Folder sub flow, at step 7; in the Edit Folder and Move Folder sub flow, at step 5, the users have not specified sufficient information to add a new Folder, the system will prompt the users for the missing information. The users can either enter the missing information or choose to cancel the operation, at which point the Use Case ends.

(iii) The users do not have the permission to view

If, in the Basic Flow at step 2, the users do not have the right to view a folder. The folder doesn't appear in the list of all files and folders.

(iv) The users do not have the permission to write

If, in the Edit Folder, Move Folder and Delete Folder sub flow, at step 5, the users do not have the right to write folder. The users can choose to cancel the operation, at which point the Use Case ends.

Special Requirement

None.

Pre-conditions

The users must be logged into the system before this Use Case begins.

Post-conditions

If the Use Case was successful, the Folder information is created, updated, deleted from the system. Otherwise, the system state is unchanged.

3.4.6 Use case: Manage Files (FEATURE 5)

Brief description

The Use Case allows the system users to manage files' information. This includes upload a new file to the system, changing existing files' information on the system, removing a file from the system, and viewing a file.

Flow of events

1. Basic Flow The table 3.13 describe when the users wish to upload, change, and/or delete File information in the system.

Actor Action	System Action	Data
1. The users want to manage files.	2. The system shows the list of all folders and files details.	
3. The users view the folders and files lists and/or filter them by custom criteria.	4. The system displays the files and folders by custom criteria.	List of all files and folders with custom criteria.
	5. The system requests that the users specify the function he/she would like to perform (either Create a File, Update File, Move File, Download File, or Delete File).	- Manage selection
6. The users select "Upload File".	7. The system requests the users to enter the File information.	- File(s) Data* - Source* - Topics* * Required fields
8. The users provide the requested information.	9. The system generates and assigns a unique File Id number to the User. The File is uploaded to the system. The User is recognized as the File's owner by the system with read and write permission. Permission for Admin is also automatically uploaded.	

Table 3.13: Manage Files Basic Flow (SUB-FEATURE 5.1 & SUB-FEATURE 5.2)

*Edit File Sub Flow*At step 6 of **Basic Flow**

Actor Action	System Action	Data
1. The users select "Update File".	2. The system requests the users to select File from File list or File details interface.	
3. The users select File.	4. The system retrieves and displays the File.	
5. The users enter the desired changes to the File information.	6. The system updates the File record with the updated information.	- Name - Source - Topics

Table 3.14: Update File Sub Flow (SUB-FEATURE 5.3)

*Add a File to a Folder Sub Flow*At step 6 of **Basic Flow**

Actor Action	System Action	Data
1. The users select "Move File".	2. The system requests the users to select File from File list or File details interface.	
3. The users select File.	4. The system retrieves and displays the File.	
5. The users enter the destination folder name.	6. The system updates the File record with the updated information.	Destina- tion File name

Table 3.15: Add a File to a Folder Sub Flow (SUB-FEATURE 5.4)

*Download File Sub Flow*At step 6 of **Basic Flow**

Actor Action	System Action	Data
1. The users select "Download File".	2. The system requests the users to select File from files and folders list or File details interface.	
3. The users select File.	4. The system retrieves and displays the File.	
5. The users select to download File.	6. The system return the File data.	File Data

Table 3.16: Download File Sub Flow (SUB-FEATURE 5.5)

*Delete File Sub Flow*At step 6 of **Basic Flow**

Actor Action	System Action	Data
1. The users selects "Delete File".	2. The system requests the users to select File from File list or File details interface.	
3. The users selects File.	4. The system retrieves and displays the File.	
5. The users select delete.	6. The system removes the File from the system.	

Table 3.17: Delete File Sub Flow (SUB-FEATURE 5.6)

2. Alternative Flows

(i) File not found

If in the Update File sub flow, at step 3 or Delete File sub flow, at step 3, a File with the specified id number does not exist, the system displays an error message. The users can cancel the operation, at which point the Use Case ends.

(ii) Invalid Format or Insufficient Information

If, in the Upload File sub flow, at step 7; in the Edit File and Move File sub flow, at step 5, the users have not specified sufficient information to upload new File, the system will prompt the users for the missing information. The users can either enter the missing information or choose to cancel the operation, at which point the Use Case ends.

- (iii) The users do not have the permission to view
If, in the Basic Flow at step 2, and in the Download File sub flow at step 5, the users do not have the right to view a file. The file doesn't appear in the list of all files and folders. And the users can't download the file.
- (iv) The users do not have the permission to write
If, in the Edit File, Move a File to a Folder and Delete File sub flow, at step 5, the users do not have the right to write folder. The users can choose to cancel the operation, at which point the Use Case ends.

Special Requirement

None.

Pre-conditions

The users must be logged into the system before this Use Case begins.

Post-conditions

If the Use Case was successful, the File information is created, updated, deleted from the system. Otherwise, the system state is unchanged.

3.4.7 Use case: Manage ACLs (FEATURE 6)

Brief description

The Use Case allows the system users to manage ACLs' information. This includes adding a new ACL to the system, changing existing ACLs' information on the system, removing a ACL from the system, and viewing a ACL.

Flow of events

1. Basic Flow The table 3.18 describe when the users wish to add, change, and/or delete ACL information in the system.

Actor Action	System Action	Data
1. The users want to manage ACLs.	2. The system shows the list of all ACLs details of the selected file or folder.	
	3. The system requests that the users specify the function he/she would like to perform (either Grant a ACL, or Delete ACL).	- Manage selection
4. The users select "Grant ACL".	5. The system requests the users to enter the ACL information.	- Source Name* - Target Id* - Source Type (File or Folder)* - Target Type (User or Group)* - Permission (Read or Write)* * Required fields
6. The users provide the requested information.	7. The system generates and assigns a ACL permission to the source. The ACL permission is added to the system.	

Table 3.18: Manage ACLs Basic Flow (SUB-FEATURE 6.1 & SUB-FEATURE 6.2)

*Delete ACL Sub Flow*At step 3 of **Basic Flow**

Actor Action	System Action	Data
1. The users selects "Delete ACL".	2. The system requests the users to select ACL from ACL list.	
3. The users select delete.	4. The system removes the ACL from the system.	

Table 3.19: Delete ACL Sub Flow (SUB-FEATURE 6.3)

2. Alternative Flows

(i) ACL not found

If in Delete ACL sub flow, at step 3, a ACL with the specified id number does not exist, the system displays an error message. The users can cancel the operation, at which point the Use Case ends.

(ii) The users do not have the permission to view

If, in the Basic Flow at step 2, the users do not have the right to view a file or a folder. They can not know the file or folder permission details.

(iii) The users do not have the permission to write

If, in the Delete sub flow, at step 2, the users do not have the right to write the selected file or folder. The users can choose to cancel the operation, at which point the Use Case ends.

(iv) Invalid Format or Insufficient Information

If, in the Grant ACL sub flow, at step 5; the users have not specified sufficient information to add a new ACL, the system will prompt the users for the missing information. The users can either enter the missing information or choose to cancel the operation, at which point the Use Case ends.

Special Requirement

None.

Pre-conditions

The users must be logged into the system before this Use Case begins.

Post-conditions

If the Use Case was successful, the ACL information is created or deleted from the system. Otherwise, the system state is unchanged.

Chapter 4

Methodology

In this section, we will list all the tools and techniques used in this project, the reasons why they are chosen with overall system architecture and the detailed use cases implementations.

4.1 System Architecture

Here, we will continue from the figure 1.1 as stated in the Introduction, zooming in to each container and component to see the details of the system architecture.

Level 2: Container diagram This is the Container diagram for ComLake Sytem. It shows that the application is made up of 3 containers: a web application Dashboard, a server-side API Application, and a Database. The Dashboard is an ReactJS application that runs in the user's web browser, providing all of the ComLake features. The Dashboard Web Application uses a JSON/HTTP API, which is provided by another Java/Spring MVC application running on the server. The Backend API Application gets user information and other information from the Database (MySQL).

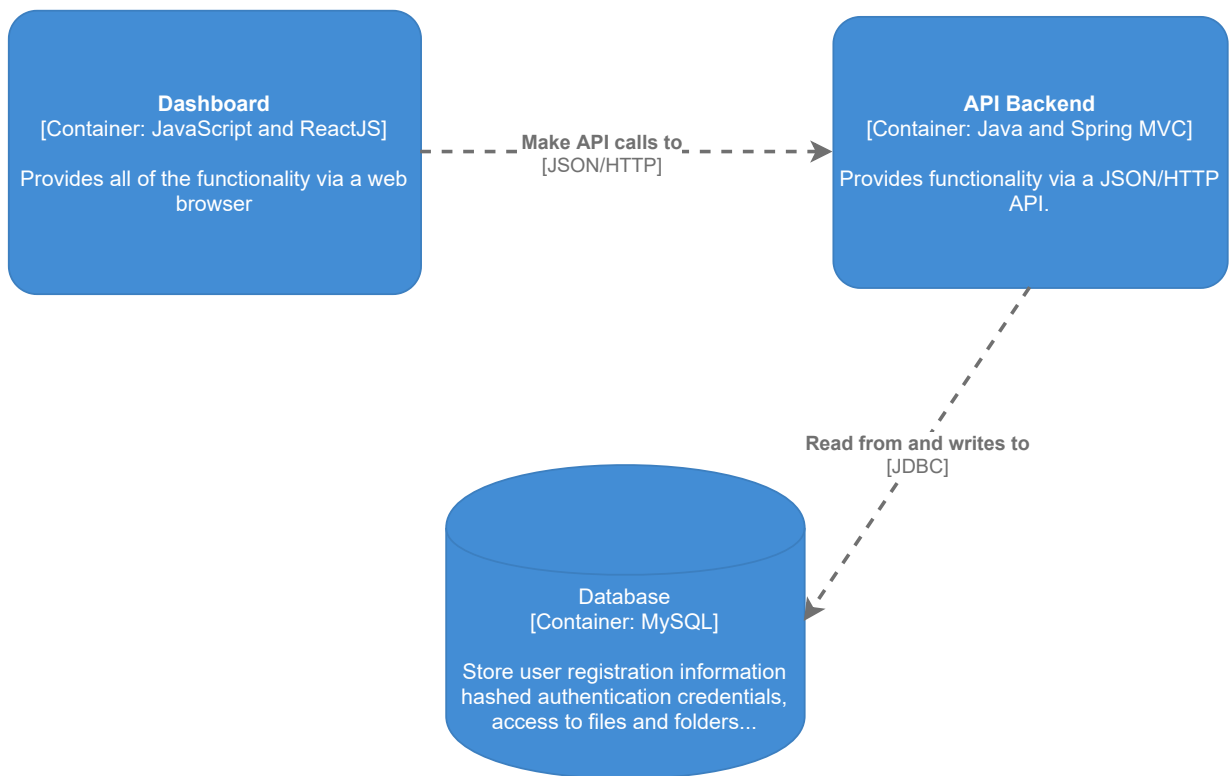


Figure 4.1: Container diagram for ComLake System - Backend API and Dashboard

Level 3: Component diagram This is the Component diagram for ComLake system, showing some of the components within the Backend API application. Here, there are six Spring MVC Rest Controllers providing access points for the JSON/HTTP API, with each controller subsequently using other components to access data from the Database.

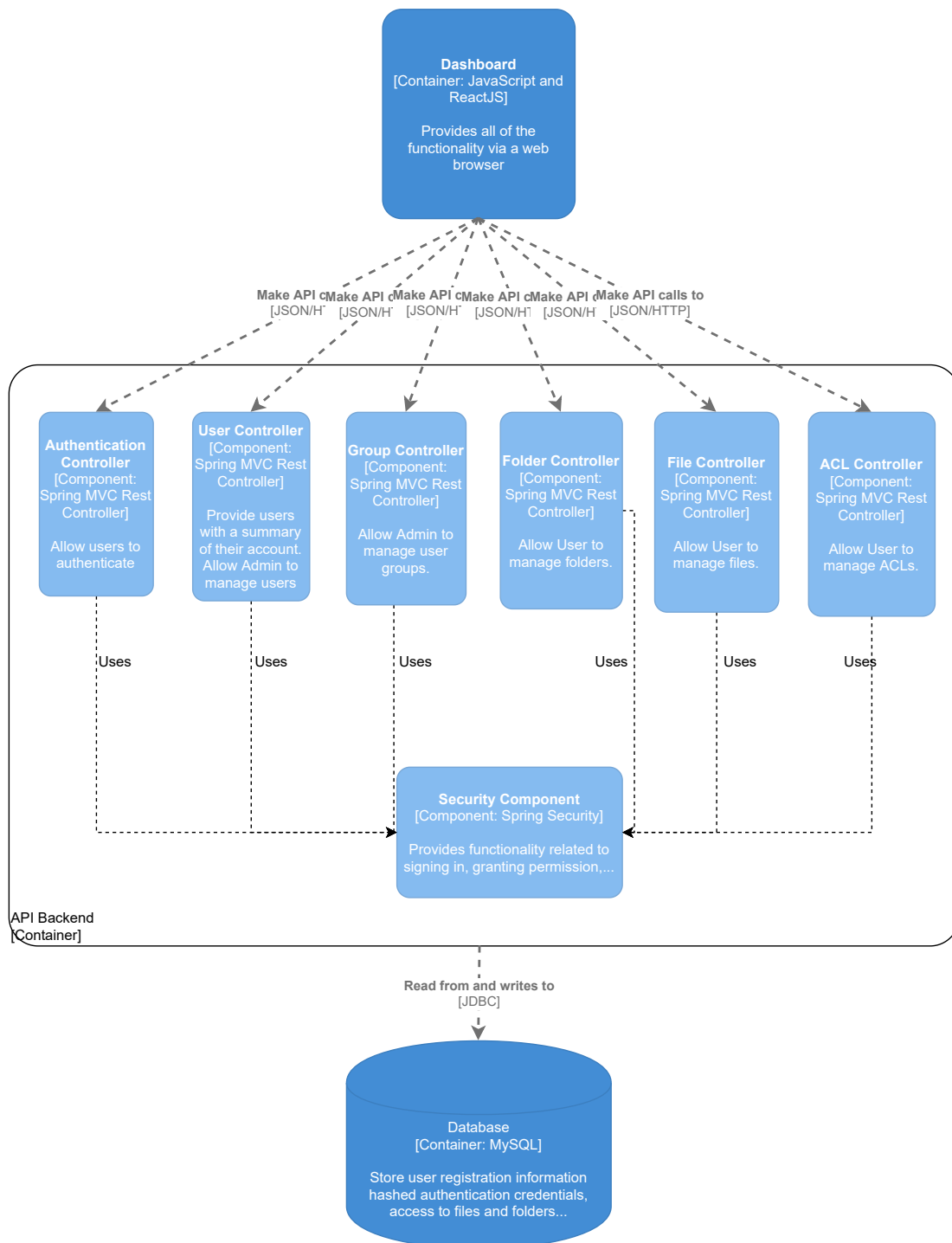


Figure 4.2: Component diagram for ComLake Sytem - Backend API

Level 4: Code Finally, this is how the API Backend is implemented as code. This package contains classes for major processing functionality within the system. Control classes exist to support authentication management, user management, user group management,

permission management, file and folder management.

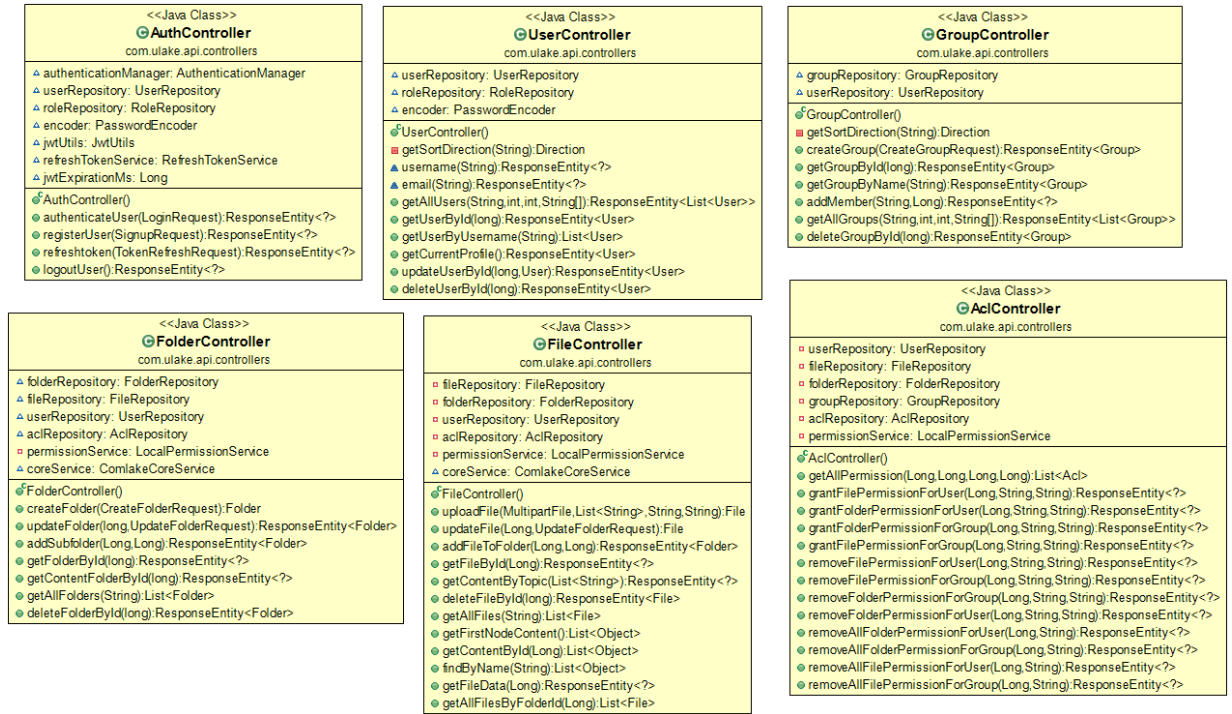


Figure 4.3: API Backend Application Package Class diagram

4.2 Database Design

Every table has the prefix "clake" to distinguish different services of the ComLake service from each other.

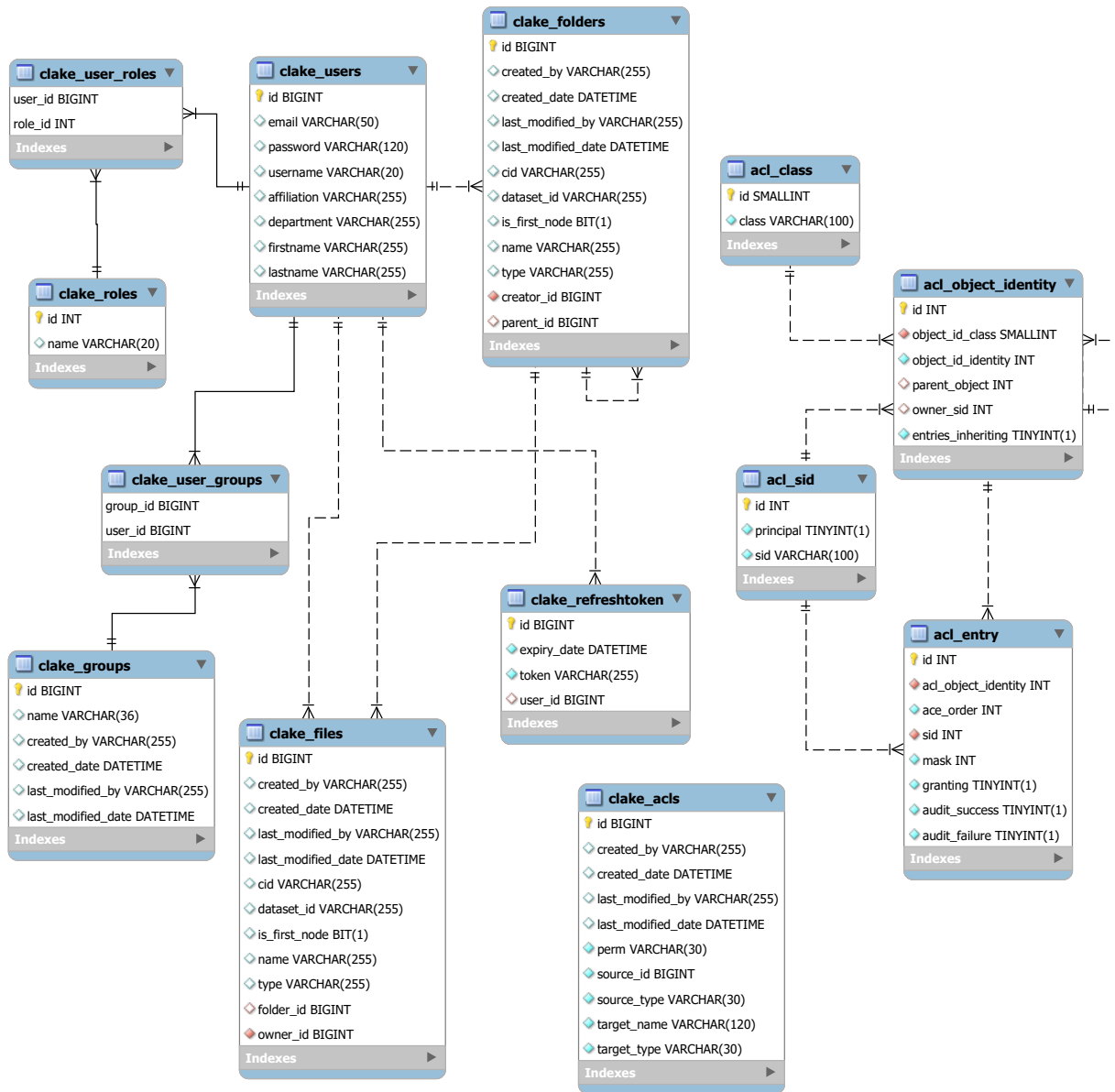
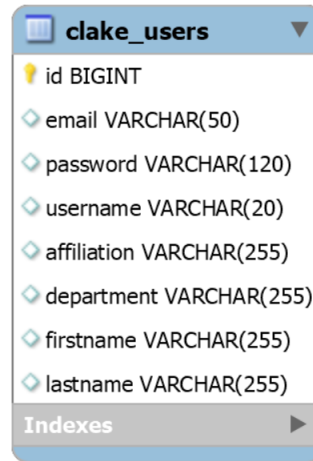


Figure 4.4: Overall Database Design

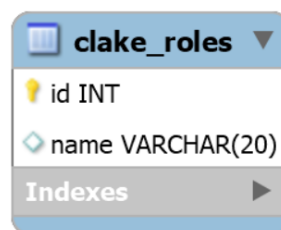
Table 4.1: Database User Design



clake_users	
id	BIGINT
email	VARCHAR(50)
password	VARCHAR(120)
username	VARCHAR(20)
affiliation	VARCHAR(255)
department	VARCHAR(255)
firstname	VARCHAR(255)
lastname	VARCHAR(255)
Indexes	

- "clake_users" is the table that contains the information of accounts. This table has sensitive data of users such as "password".
- The 'password' field in this table is saved in an encrypted form not easily used by attackers.
- After the user register, the system saves the user information and assign a unique "id" to the user, and this "id" is the Primary key of the table.

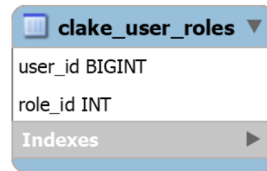
Table 4.2: Database Role Design



clake_roles	
id	INT
name	VARCHAR(20)
Indexes	

- "clake_roles" is the table that contains the information of roles, for example role name.
- In this system, we only have 2 role. Role Admin and role (Regular) User.
- This table has "id" as Primary Key. Each role has an unique "id".

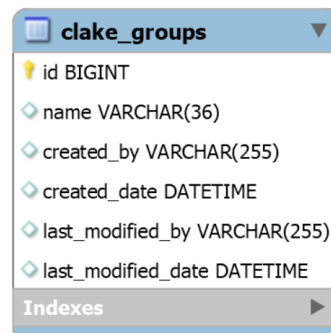
Table 4.3: Database User Role Design



clake_user_roles	
user_id	BIGINT
role_id	INT
Indexes	

- "clake_user_roles" is a table containing user and their associated role.
- This table connects with the "clake_users" table and "clake_roles" table with the Foreign Key: 'user_id' and 'role_id'. This is "many to many" relationship: Each "User" could have many roles. Each "Role" could have many users.

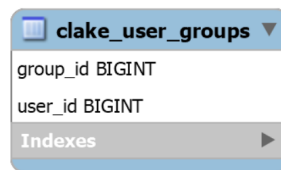
Table 4.4: Database Group Design



clake_groups	
id	BIGINT
name	VARCHAR(36)
created_by	VARCHAR(255)
created_date	DATETIME
last_modified_by	VARCHAR(255)
last_modified_date	DATETIME
Indexes	

- "clake_groups" is the table that contains the information of groups, for example group name.
- This table has "id" as Primary Key. Each group has an unique "id".
- Group name must be unique from each other.
- This table is audit-able, by the field 'created_by', 'created_date', 'last_modified_by' and 'last_modified_date'. It will monitoring and recording of selected user database actions.

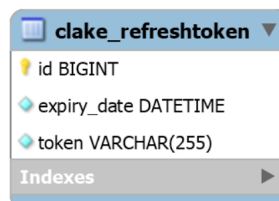
Table 4.5: Database User Group Design



clake_user_groups ▼	
group_id	BIGINT
user_id	BIGINT
Indexes ▶	

- "clake_user_groups" is a table containing user and their associated group.
- This table connects with the "clake_users" table and "clake_groups" table with the Foreign Key: 'user_id' and 'group_id'. This is "many to many" relationship: Each "User" could have many groups. Each "Group" could have many users.

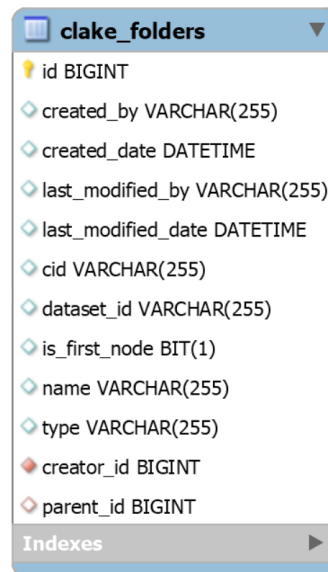
Table 4.6: Database Refresh Token Design



clake_refresh_token ▼	
id	BIGINT
expiry_date	DATETIME
token	VARCHAR(255)
Indexes ▶	

- "clake_refresh_token" is the table that contains the refresh token information of an user.
- This will keep user logged in longer time with silent authentication because it has longer expiry date than the access token.
- This table has "id" as Primary Key.
- This has one-to-one relationship with "User".

Table 4.7: Database Folder Design



clake_folders	
id	BIGINT
created_by	VARCHAR(255)
created_date	DATETIME
last_modified_by	VARCHAR(255)
last_modified_date	DATETIME
cid	VARCHAR(255)
dataset_id	VARCHAR(255)
is_first_node	BIT(1)
name	VARCHAR(255)
type	VARCHAR(255)
creator_id	BIGINT
parent_id	BIGINT
Indexes	

- "clake_folders" is the table that contains the information of folders.
- This table has "id" as Primary Key. Each folder has an unique "id".
- This table connects with the "clake_users" table with the Foreign Key 'creator_id'. This is "many to one" relationship: Many "Folder" belongs to a "User".
- The column 'parent_id' is a foreign key that refers to the primary key column 'id' of the table itself. This is parent-child relationship, it needed to represent nested, hierarchical structures: A "Folder" can have one or many children folders below it. And vice-versa, a "Folder" can have one or many parent folders above it. There is no limit for the nested level.
- Noted that this table doesn't include 'source' and 'topics' fields because that meta-data is stored in the ComLake Core. We use 'cid' and 'dataset_id' as identifier to connect to the Core.
- This table is audit-able, by the field 'created_by', 'created_date', 'last_modified_by' and 'last_modified_date'. It will monitoring and recording of selected user database actions.

Table 4.8: Database File Design

clake_files	
id	BIGINT
created_by	VARCHAR(255)
created_date	DATETIME
last_modified_by	VARCHAR(255)
last_modified_date	DATETIME
cid	VARCHAR(255)
dataset_id	VARCHAR(255)
is_first_node	BIT(1)
name	VARCHAR(255)
type	VARCHAR(255)
folder_id	BIGINT
owner_id	BIGINT
Indexes	

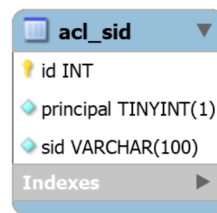
- "clake_files" is the table that contains the information of files.
- This table has "id" as Primary Key. Each file has an unique "id".
- This table connects with the "clake_users" table with the Foreign Key 'owner_id'. This is "many to one" relationship: Many "File" belongs to a "User".
- This table connects with the "clake_folders" table with the Foreign Key 'folder_id'. This is "many to one" relationship: Many "File" could be in a "Folder".
- Noted that this table doesn't include 'source' and 'topics' fields because that meta-data is stored in the ComLake Core. We use 'cid' and 'dataset_id' as identifier to connect to the Core.
- This table is audit-able, by the field 'created_by', 'created_date', 'last_modified_by' and 'last_modified_date'. It will monitoring and recording of selected user database actions.

Table 4.9: Database ACL Class Design

acl_class	
id	SMALLINT
class	VARCHAR(100)
Indexes	

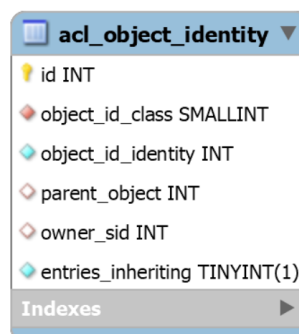
- "acl_class" is the table that stores the class name of the domain object.
- This table has "id" as Primary Key. Each file has an unique "id".
- In this system, we only have two classes that need to be secured: "File" and "Folder". So it will store two values: "com.ulake.api.models.File" and "com.ulake.api.models.Folder".

Table 4.10: Database ACL Sid Design



- "acl_sid" is the table that allows us to universally identify any users, roles or groups in the system.
- This table has "id" as Primary Key. Each file has an unique "id".
- The field 'sid' could be username of the user or role name like "ROLE_ADMIN" or group name. SID stands for "Security Identity".
- The field 'principal' could be 0 or 1. It is a Boolean value indicate if the corresponding 'sid' is a principal (user) or an authority (role or group).

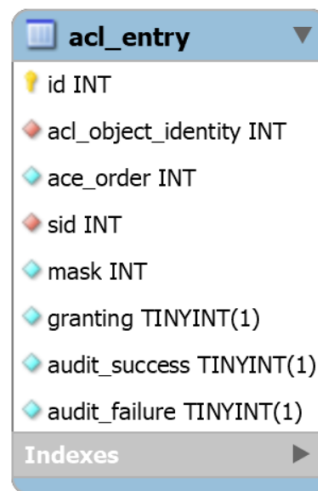
Table 4.11: Database ACL Object Identity Design



- "acl_object_identity" is the table that stores the information for each unique domain object (File or Folder).

- This table has "id" as Primary Key. Each file has an unique "id".
- The field 'object_id_class' indicate if it is File or Folder.
- The field 'owner_sid' links to "acl_sid" table. It is the Id of the owner.

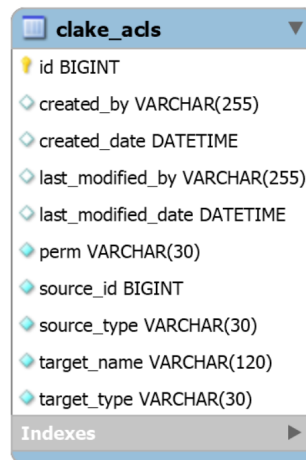
Table 4.12: Database ACL Entry



acl_entry	
id	INT
acl_object_identity	INT
ace_order	INT
sid	INT
mask	INT
granting	TINYINT(1)
audit_success	TINYINT(1)
audit_failure	TINYINT(1)
Indexes	

- "acl_entry" is the table that stores individual permission assigns to each SID on an Object Identity.
- This table has "id" as Primary Key. Each file has an unique "id".
- The field 'acl_object_identity' specify the object identity, links to "acl_object_identity" table.
- The field 'ace_order' is the order of current entry in the ACL entries list of corresponding Object Identity.
- The field 'sid' is the target SID which the permission is granted to or denied from, links to "acl_sid" table.

Table 4.13: Database ACL Design



clake_acls	
id	BIGINT
created_by	VARCHAR(255)
created_date	DATETIME
last_modified_by	VARCHAR(255)
last_modified_date	DATETIME
perm	VARCHAR(30)
source_id	BIGINT
source_type	VARCHAR(30)
target_name	VARCHAR(120)
target_type	VARCHAR(30)
Indexes	

- "clake_acls" is the table that mimics the table "acl_entry" for easier interface.
- This table has "id" as Primary Key. Each file has an unique "id".
- The field 'source_type' could be "File" or "Folder".
- The field 'target_type' could be "User" or "Group".
- The field 'perm' could be "Read" or "Write".
- This table is audit-able, by the field 'created_by', 'created_date', 'last_modified_by' and 'last_modified_date'. It will monitoring and recording of selected user database actions.

4.3 Use Cases Implementation

4.3.1 Login

This use case describes how a user logs into the system. This user can be a Admin or a (Regular) User. When a user wants to login, the system requests the actor to enter username and password (Figure 1 in Appendices). Once the user enters the username and password, the system will check if the information is valid and logs the user in. Otherwise, the system displays an error message and asks the user to re-enter the information.

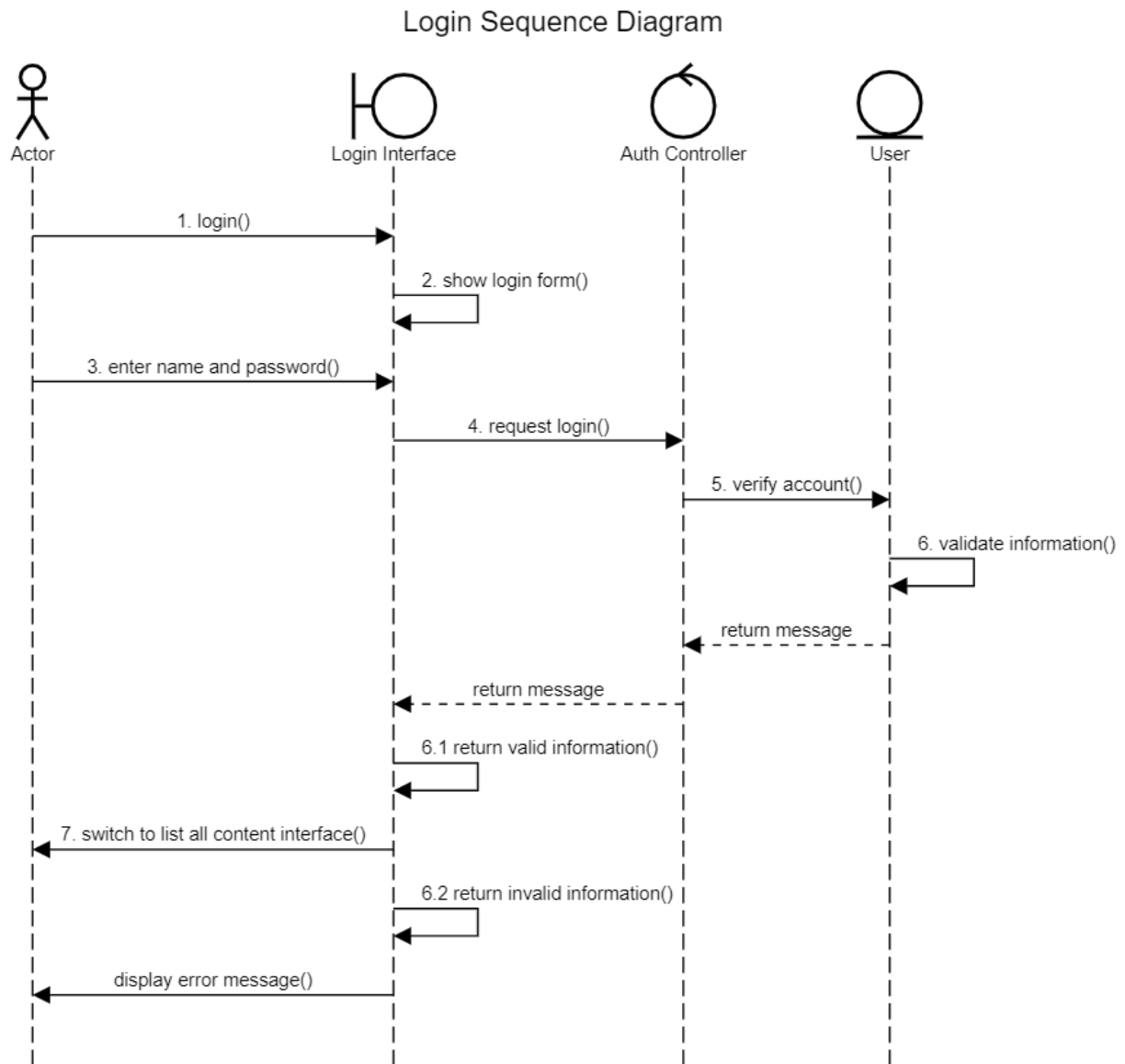


Figure 4.5: Login Sequence diagram

4.3.2 Register

This use case describes how a user registers to the system. When a user wants to register into the system, the system will display a register interface and requests that the user enter first name, last name, email, username, and password (Figure 2 in Appendices). After the user enters the required information, the system checks if the entered email and username are unique from the table "comlake.users" in Database. If unique, the system encrypts the password, saves the user information, assigns a unique id to the user, and redirects to the login interface. Otherwise, the system displays an error message and asks the user to re-enter the information.

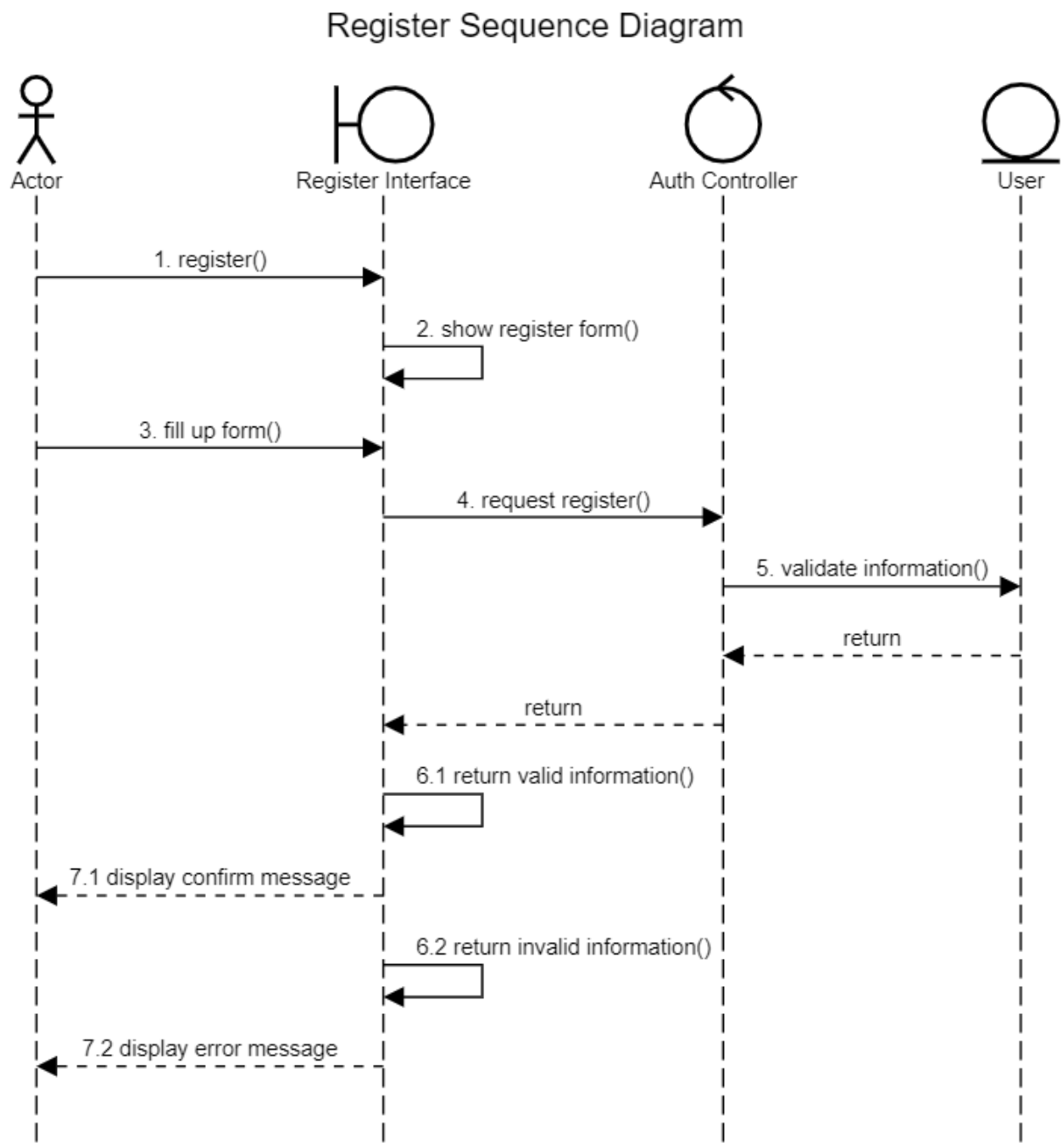


Figure 4.6: Register Sequence diagram

4.3.3 Manage Users

This Use Case describes how an Admin can manage users.

Manage Users - Basic flow

When the Admin logged into the system, there will be a User tab if the Admin wishes to manage users. The system lists all of the users (Figure 3 in Appendices). The Admin can filter users with desired criteria, and the system will respond to a list of users according to the criteria. The Admin could choose to perform an action (create, edit, delete). If the Admin chooses to create a user, there will be a create user form (Figure 4 in Appendices). After the Admin enters the required information, the system checks if the entered email and username are unique from the table "comlake_users" in Database. If unique, the system encrypts the password, saves the user information, assigns a unique id to the user, and redirects to the login interface. Otherwise, the system displays an error message and asks the Admin to re-enter the information.

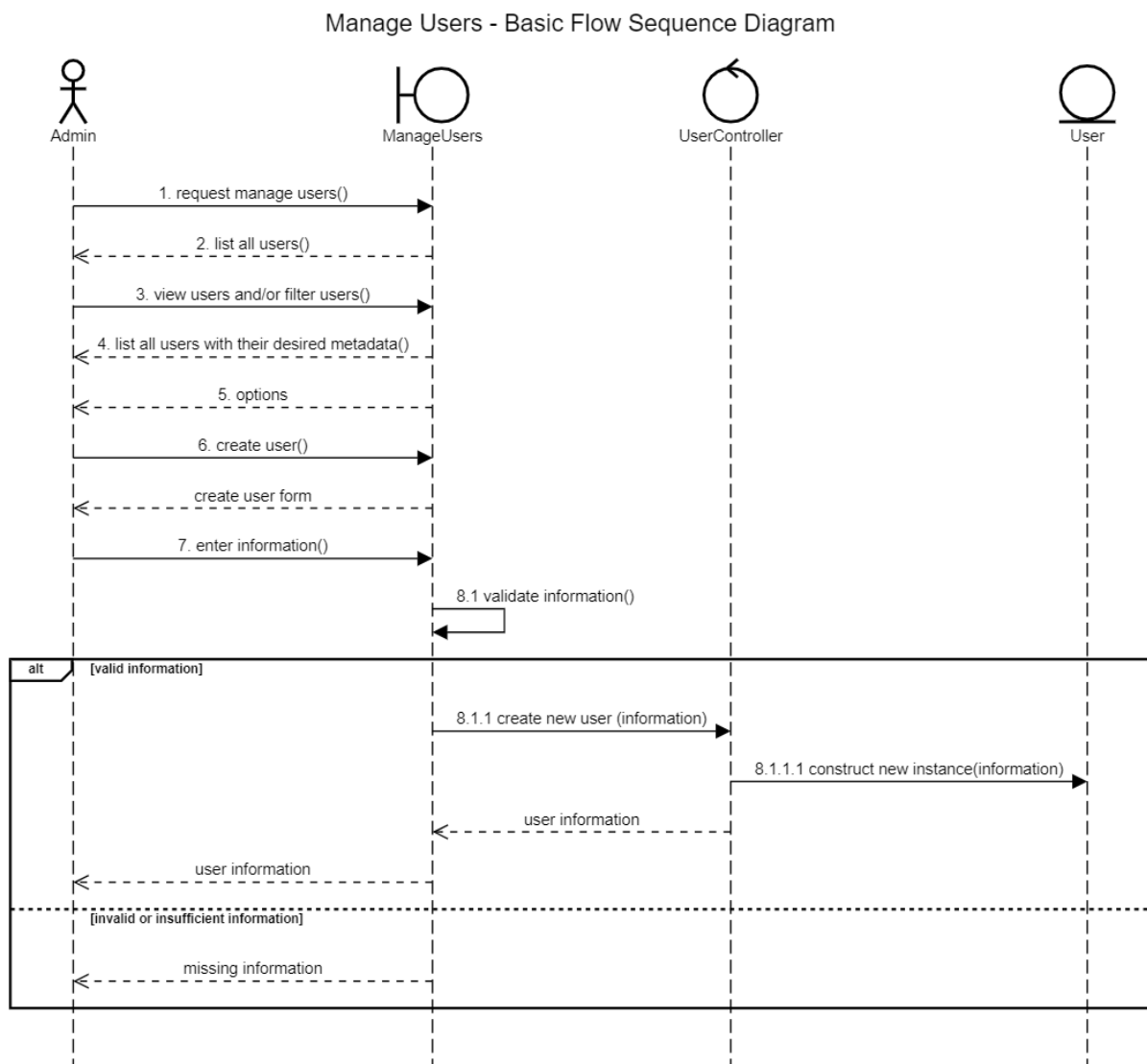


Figure 4.7: Manage Users - Basic Flow Sequence diagram

Update User - Sub flow

If the Admin wishes to edit a user by select the target user, the system will display an update user form (Figure 5 in Appendices). After the Admin enters the updated information, the system will check if the user exists or not. If the user exists, the system will save the updated information of the user. If not, the system displays an error message.

Manage Users - Update User Sub Flow Sequence Diagram

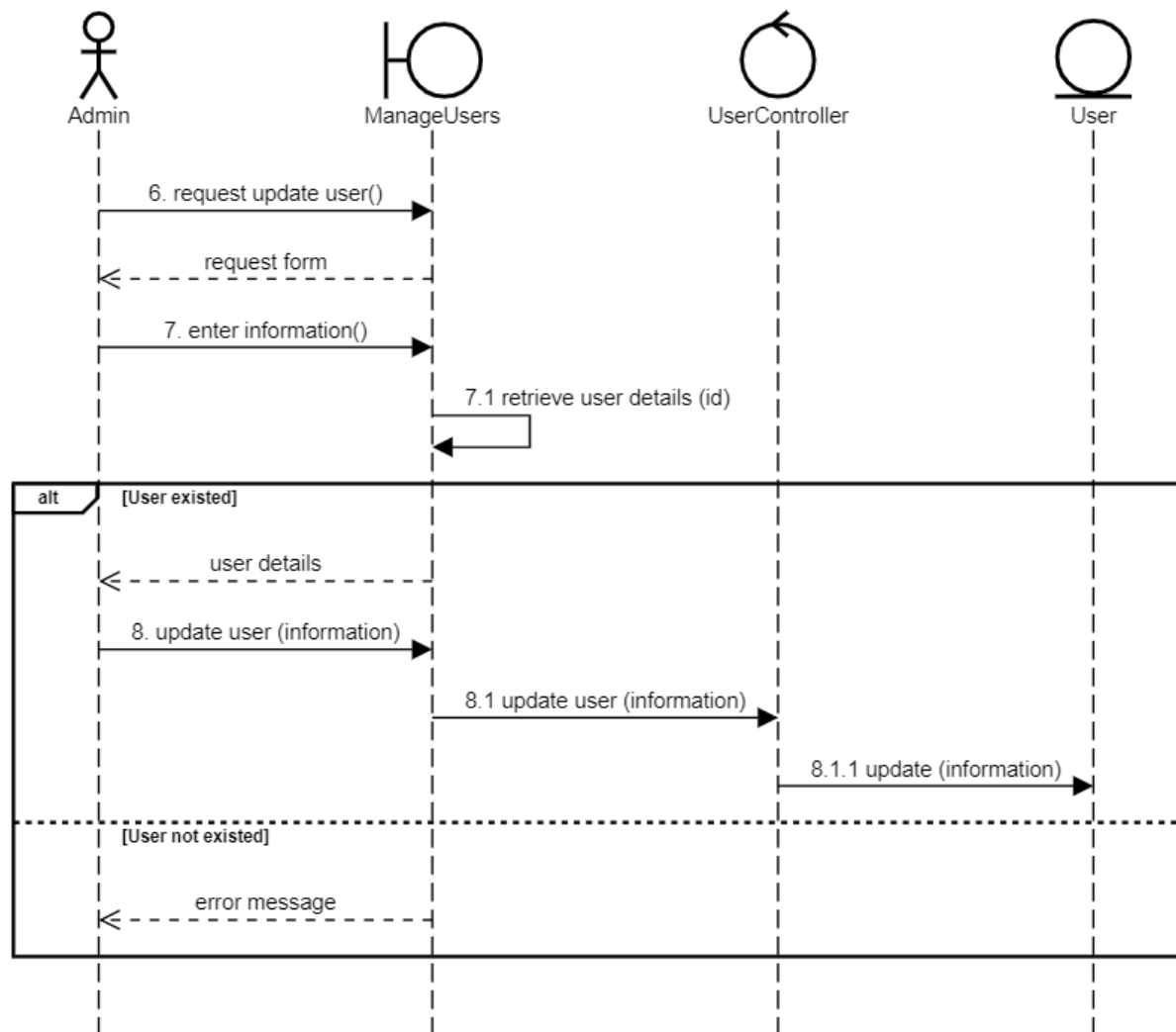


Figure 4.8: Manage Users - Update User Sub Flow Sequence diagram

Delete User - Sub flow

If the Admin wished to delete a user by selecting the target user, the system would display the full details of the user. When Admin clicks on the "Delete" button, the system will checks if the user exists or not. If the user exists, the system will remove the user. If not, the system displays an error message.

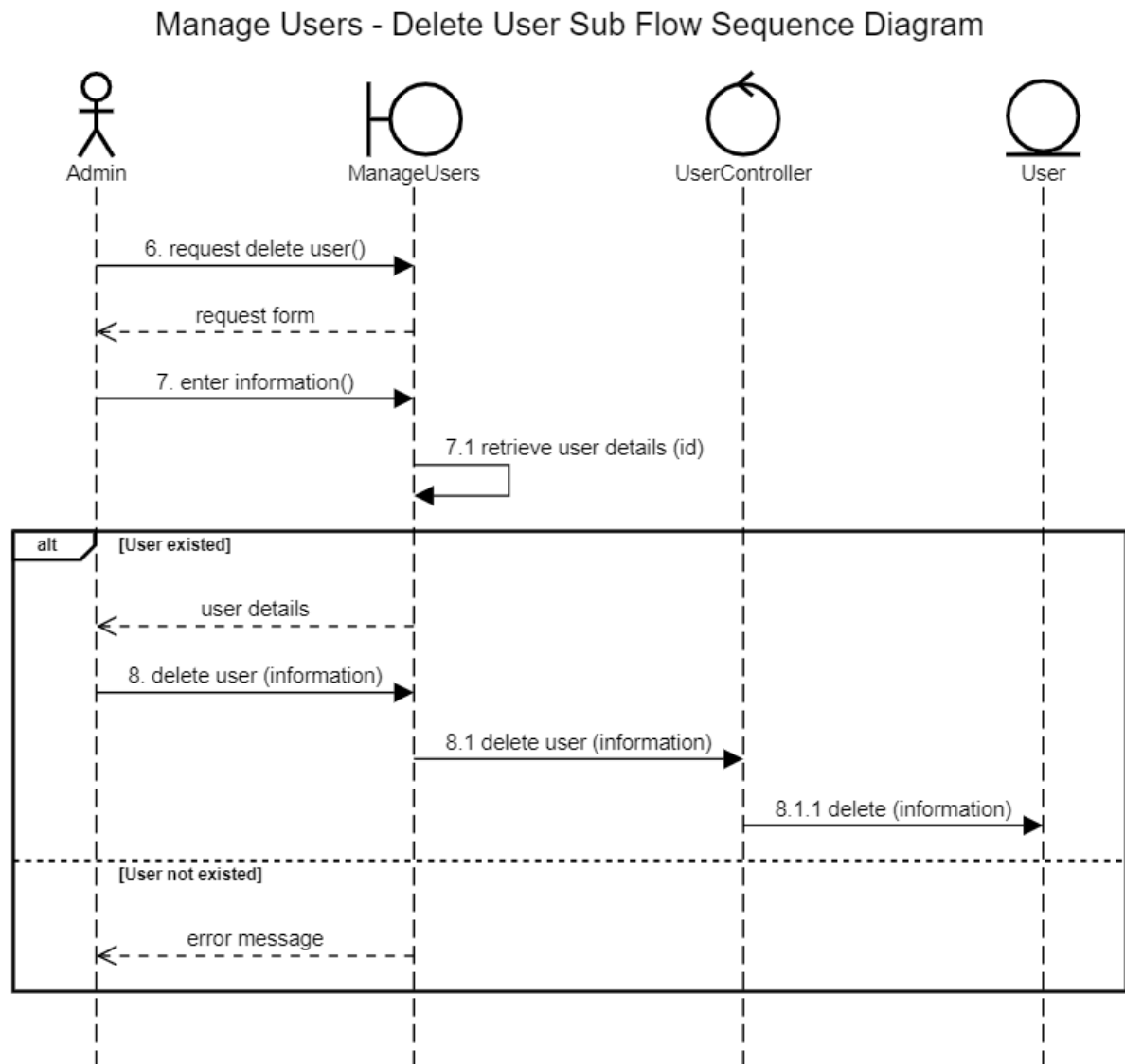


Figure 4.9: Manage Users - Delete User Sub Flow Sequence diagram

4.3.4 Manage User Groups

This Use Case describes how an Admin can manage user groups.

Manage User Groups - Basic flow

When the Admin logged into the system, there will be a User tab if the Admin wishes to manage user groups. The system lists all of the user groups (Figure 6 in Appendices). The Admin can filter user groups with desired criteria, and the system will respond to a list of user groups according to the criteria. The Admin could choose to perform action (create, edit, delete). If the Admin choose to create group, there will be a create group form (Figure

8 in Appendices). After the Admin enters the required information, the system checks if the entered name are unique from the table "comlake_user_groups" in Database. If unique, the system saves the group information, assigns a unique id to the group, and redirects to the login interface. Otherwise, the system displays an error message and asks the Admin to re-enter the information.

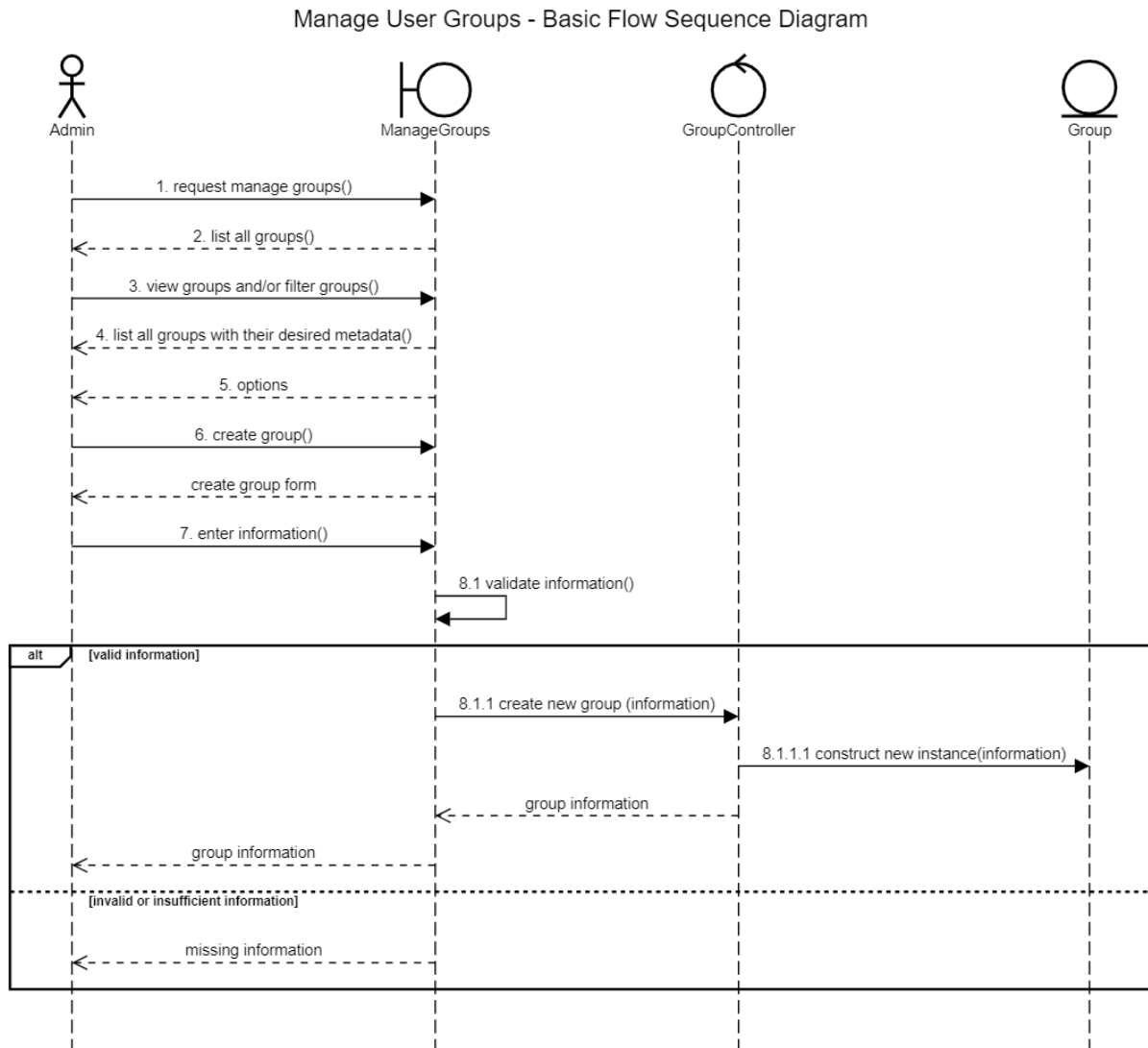


Figure 4.10: Manage Groups - Basic Flow Sequence diagram

Update Group - Sub flow

If the Admin wishes to edit by selecting the target group, the system will display an update group[form (Figure 9 in Appendices). After the Admin enters the updated information, the system will check if the group exists or not. If the group exists, the system will save the updated information of the group. If not, the system displays an error message.

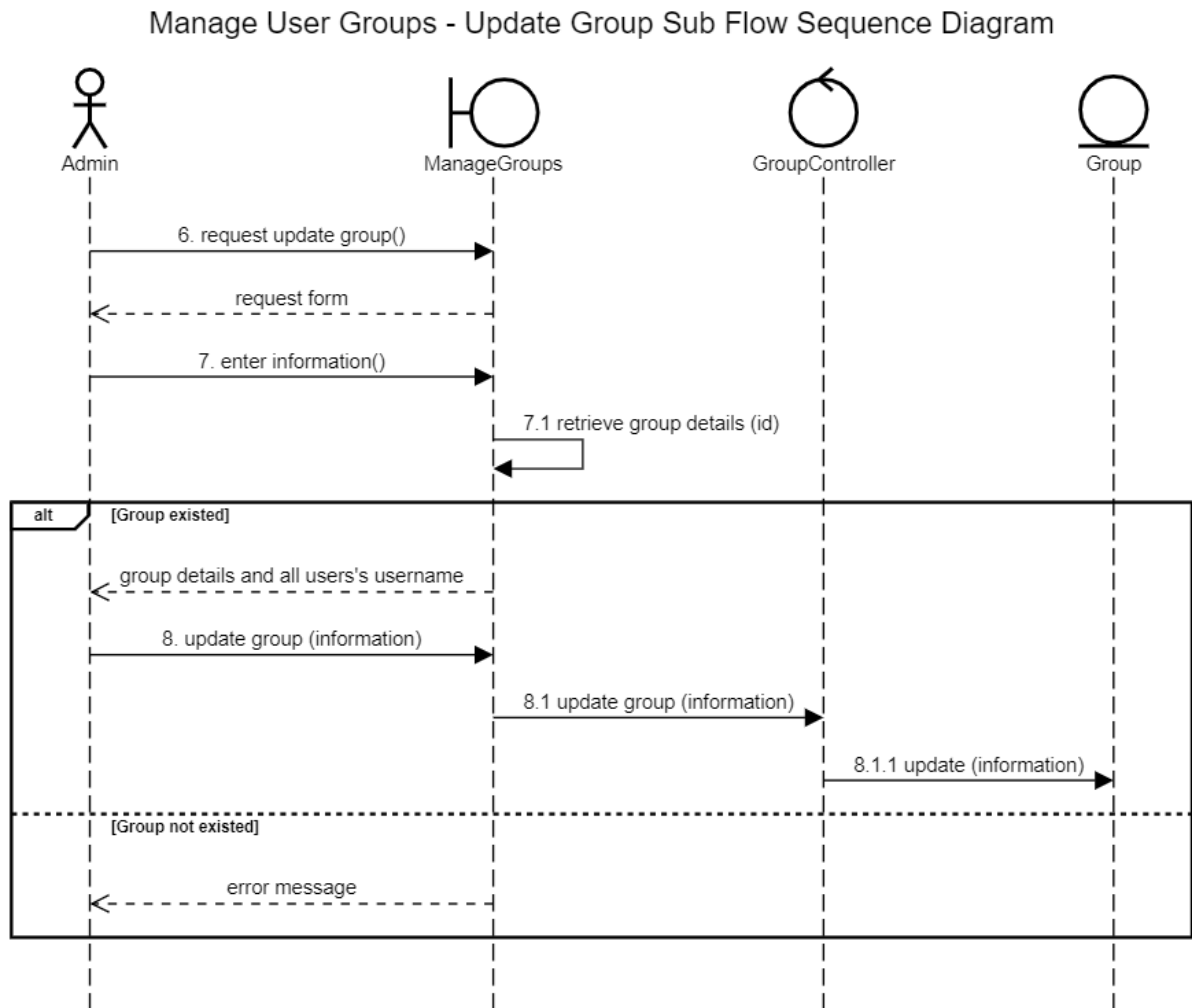


Figure 4.11: Manage User Groups - Update Group Sub Flow Sequence diagram

Delete Group - Sub flow

If the Admin wished to delete a group by selecting the target group, the system would display the full details of the group. When Admin clicks on the "Delete" button, the system will check if the group exists or not. If the group exists, the system will remove the group. If not, the system displays an error message.

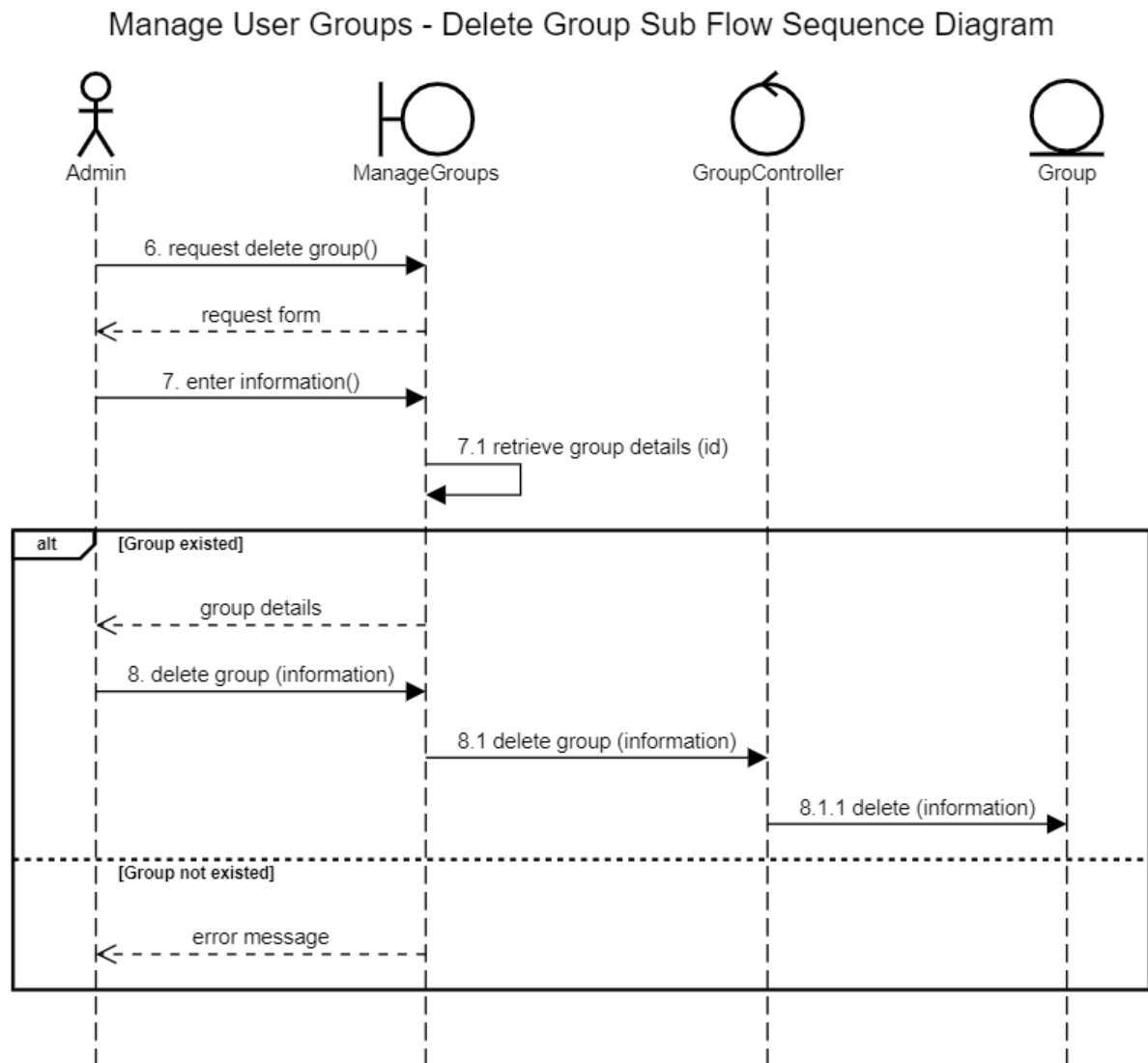


Figure 4.12: Manage User Groups - Delete Group Sub Flow Sequence diagram

4.3.5 Manage Folders

This Use Case describes how a User can manage folders.

Manage Folders - Basic flow

When the User logged into the system, the system lists all of the files and folders (Figure 5.1 in Appendices). The User can filter folders with desired criteria, and the system will respond to a list of folders. The User could choose to perform an action (create, edit, delete, move). If the User does not have permission to view a folder, that folder will not appear in the folder list. If the User chooses to create a folder, there will be a create folder form (Figure 10 in Appendices). After the User enters all the required information, the

system saves the folder information, assigns a unique id to the folder, and redirects to the files and folders interface. Otherwise, the system displays an error message and asks the User to re-enter the information.

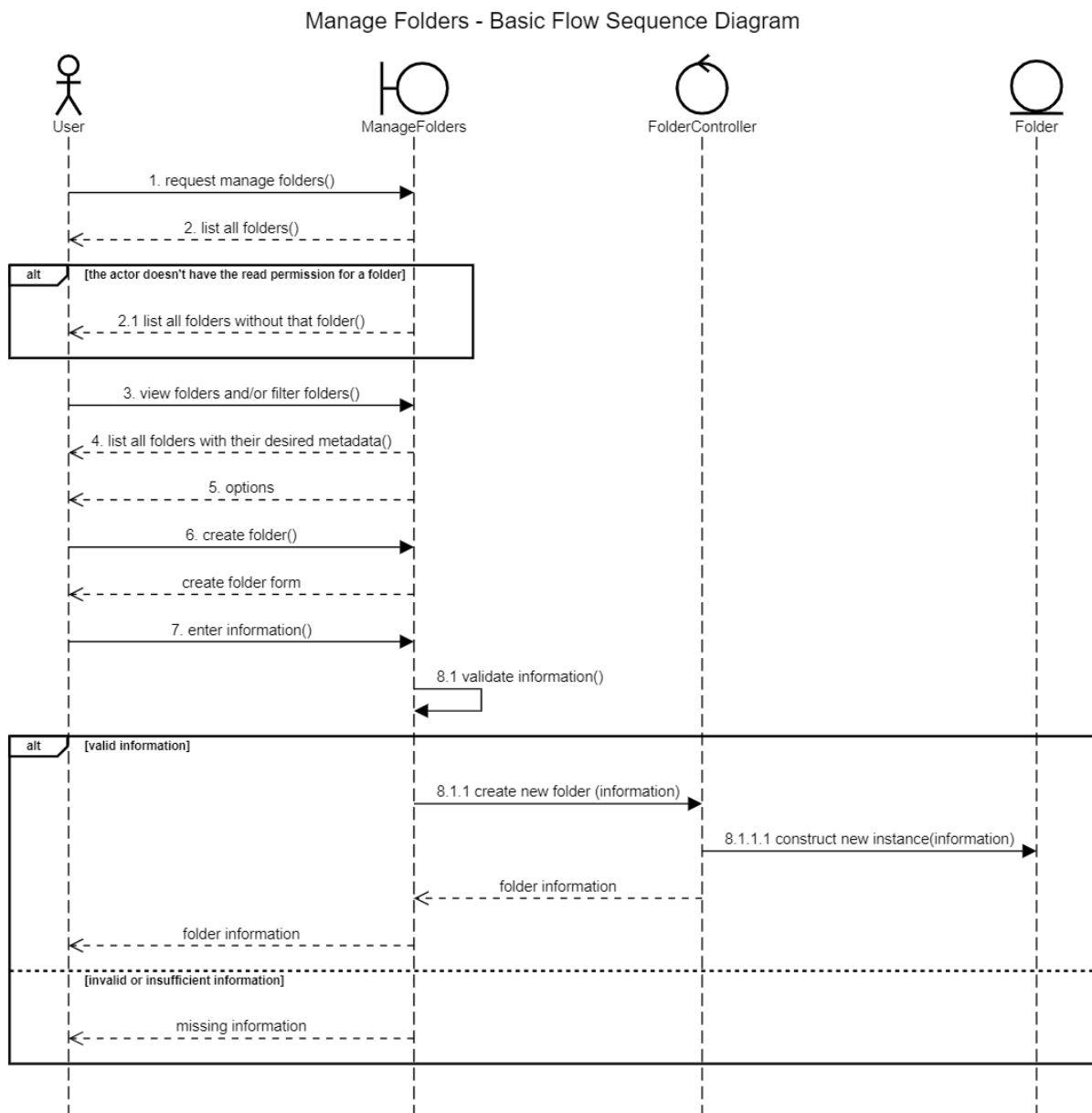


Figure 4.13: Manage Folders - Basic Flow Sequence diagram

Update Folder - Sub flow

If the User wishes to edit a folder by select the target folder, the system will check if the User have the right to write the folder. If the User does not have a write permission to a folder, the system will display en error message. Else, the system will display an update

folder f12 in Appendices). After the User enters the updated information, the system will check if the folder exists or not. If the folder exists, the system will save the updated information of the folder. If not, the system displays an error message.

Manage Folders - Update Folder Sub Flow Sequence Diagram

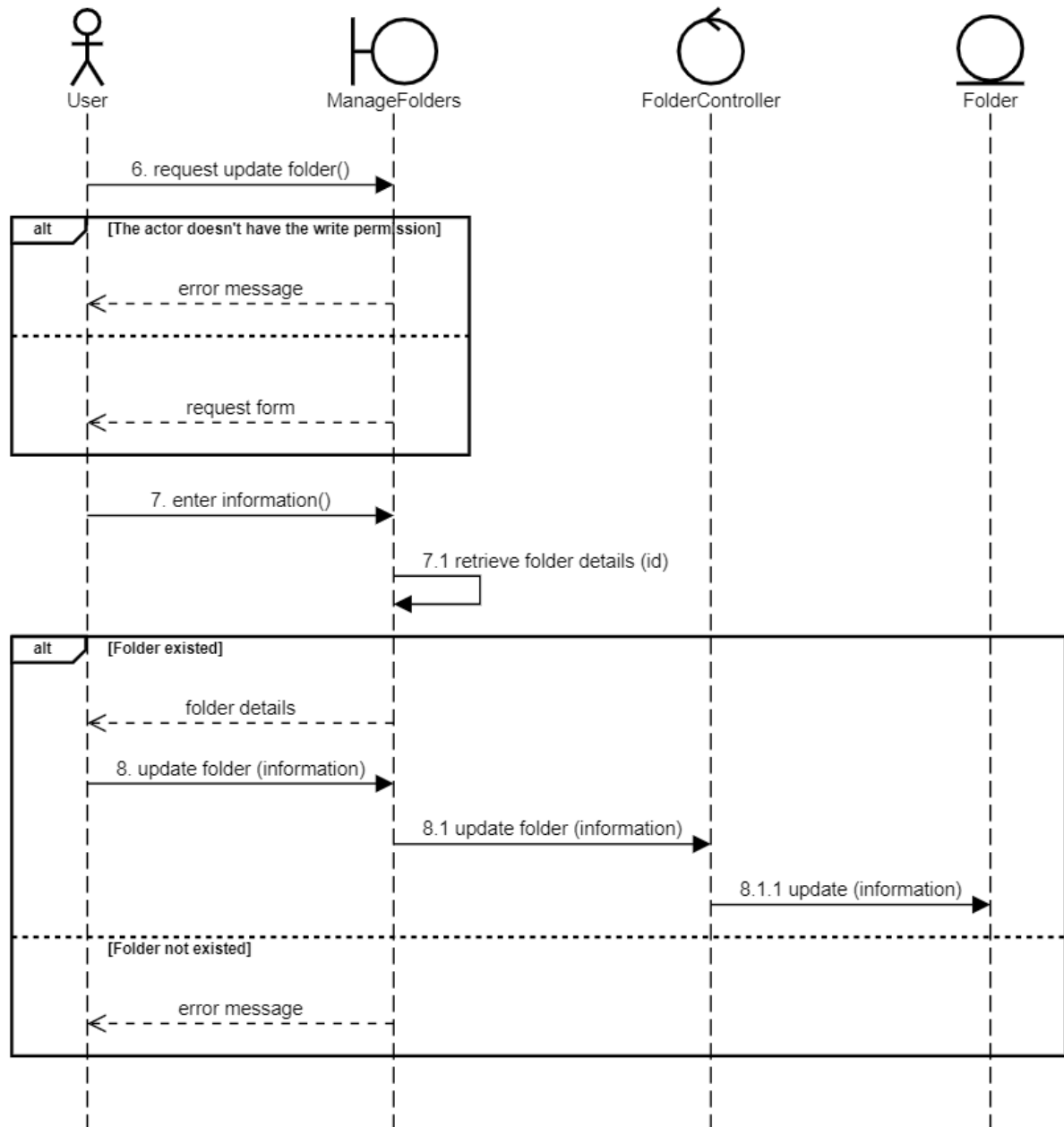


Figure 4.14: Manage Folders - Update Folder Sub Flow Sequence diagram

Delete Folder - Sub flow

If the User wished to delete a folder by selecting the target folder, the system will check if the User have the right to write the folder. If the User does not have a write permission to a folder, the system will display en error message. Else, the system would display the full details of the folder. When User clicks on the "Delete" button, the system will checks if the folder exists or not. If the folder exists, the system will remove the folder. If not, the system displays an error message.

Manage Folders - Delete Folder Sub Flow Sequence Diagram

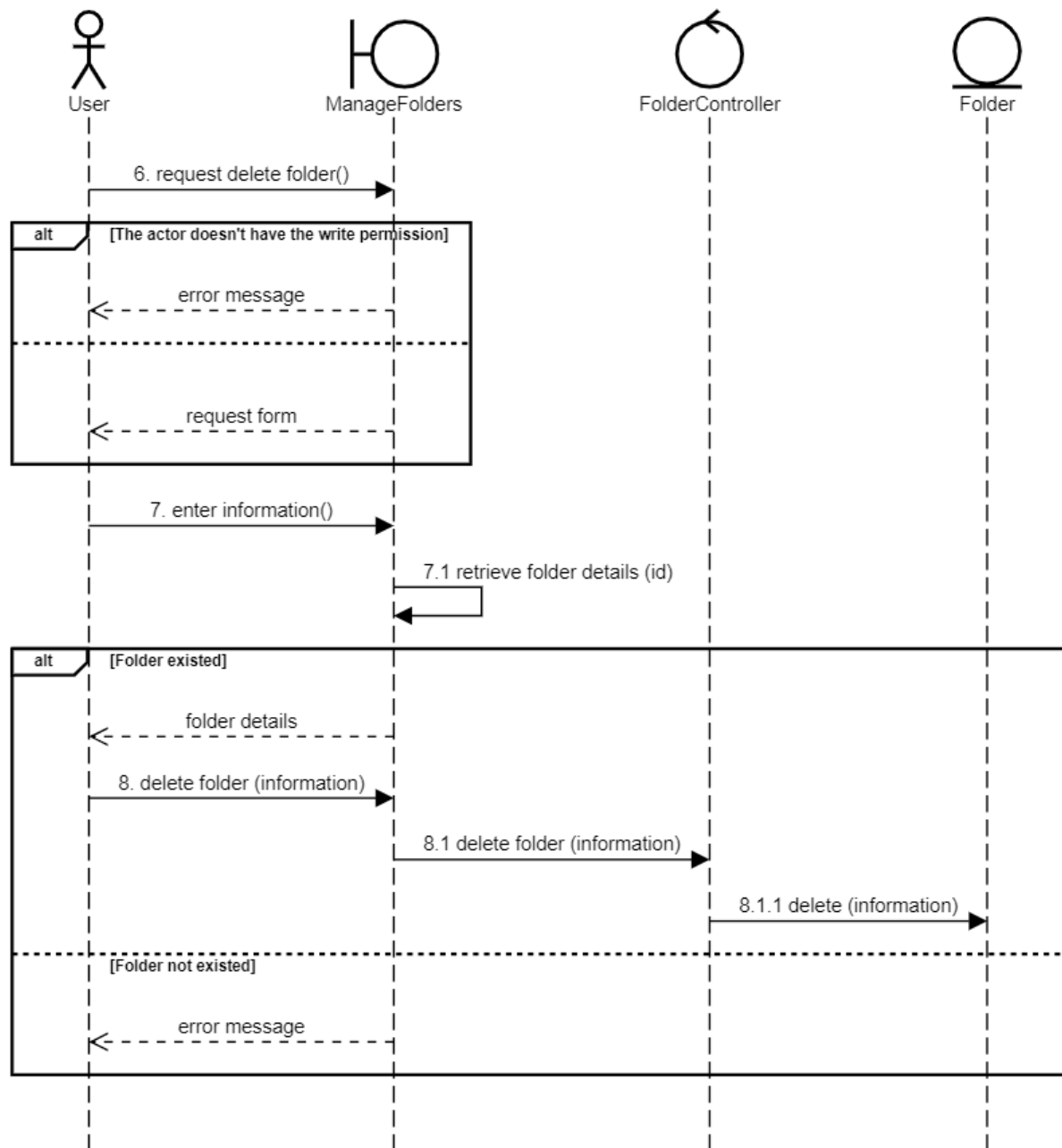


Figure 4.15: Manage Folders - Delete Folder Sub Flow Sequence diagram

Move a Sub-folder a Folder - Sub flow

If the User wishes to move a folder, the system will check if the User have the right to write the folder. If the User does not have a write permission to a folder, the system will display en error message. Else, the system will display a move folder form (Figure 13). The User will select a folder as source and enter the targeted folder and the destination

folder. The system will move the targeted folder accordingly.

Manage Folders - Move a Sub-folder to a Folder Sub Flow Sequence Diagram

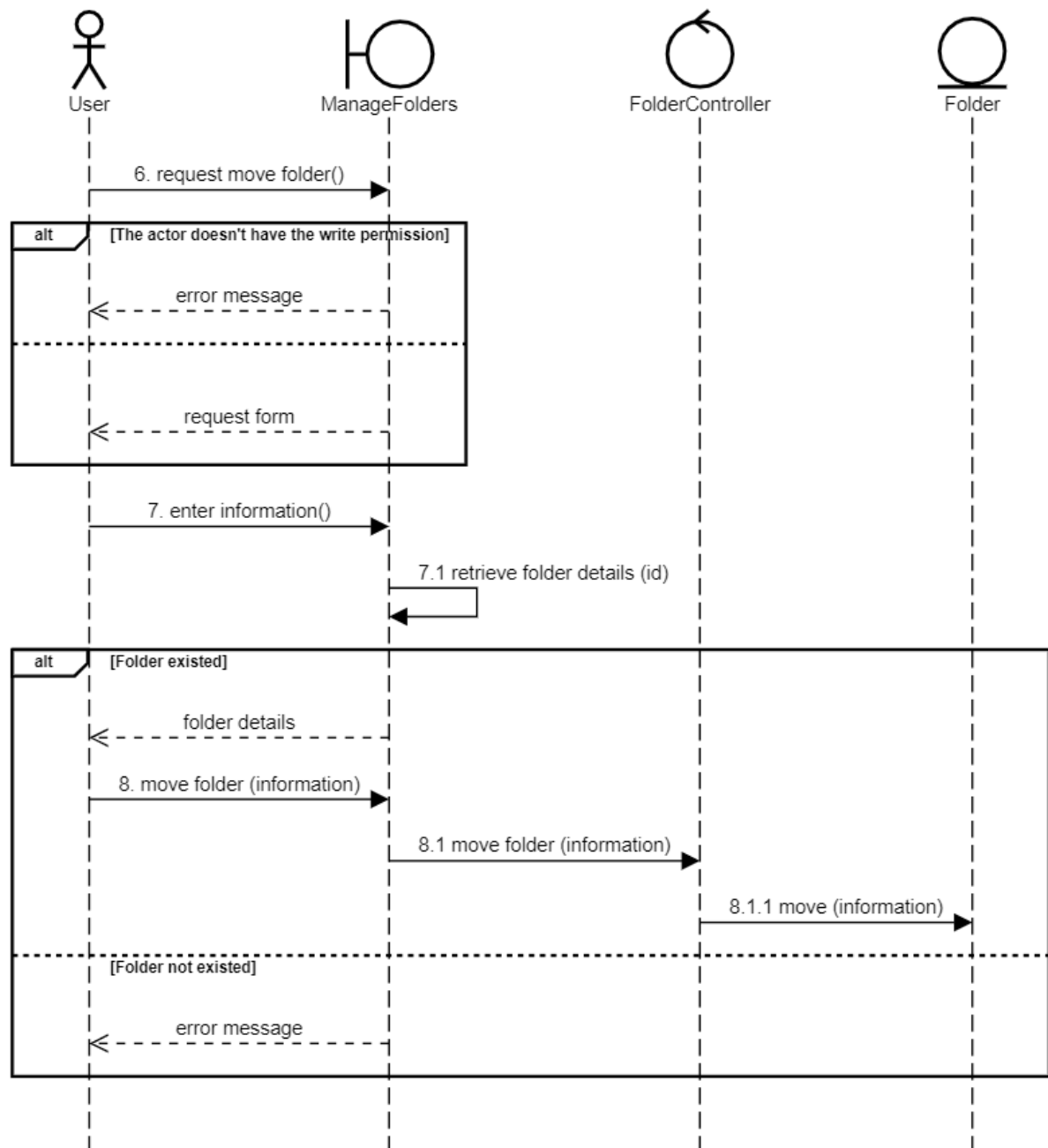


Figure 4.16: Manage Folders - Move a Sub-folder to a Folder Sub Flow Sequence diagram

4.3.6 Manage Files

This Use Case describes how a User can manage files.

Manage Files - Basic flow

When the User logged into the system, the system lists all of the files and folders (Figure 5.1 in Appendices). The User can filter files with desired criteria, and the system will respond to a list of files. If the User does not have permission to view a file, that file will not appear in the file list. The User could choose to perform an action (upload, edit, delete, move, download). If the User chooses to upload, there will be an upload file form (Figure 5.2 in Appendices). After the User enters all the required information, the system saves the file information, assigns a unique id to the file, and redirects to the files and folders interface. Otherwise, the system displays an error message and asks the User to re-enter the information.

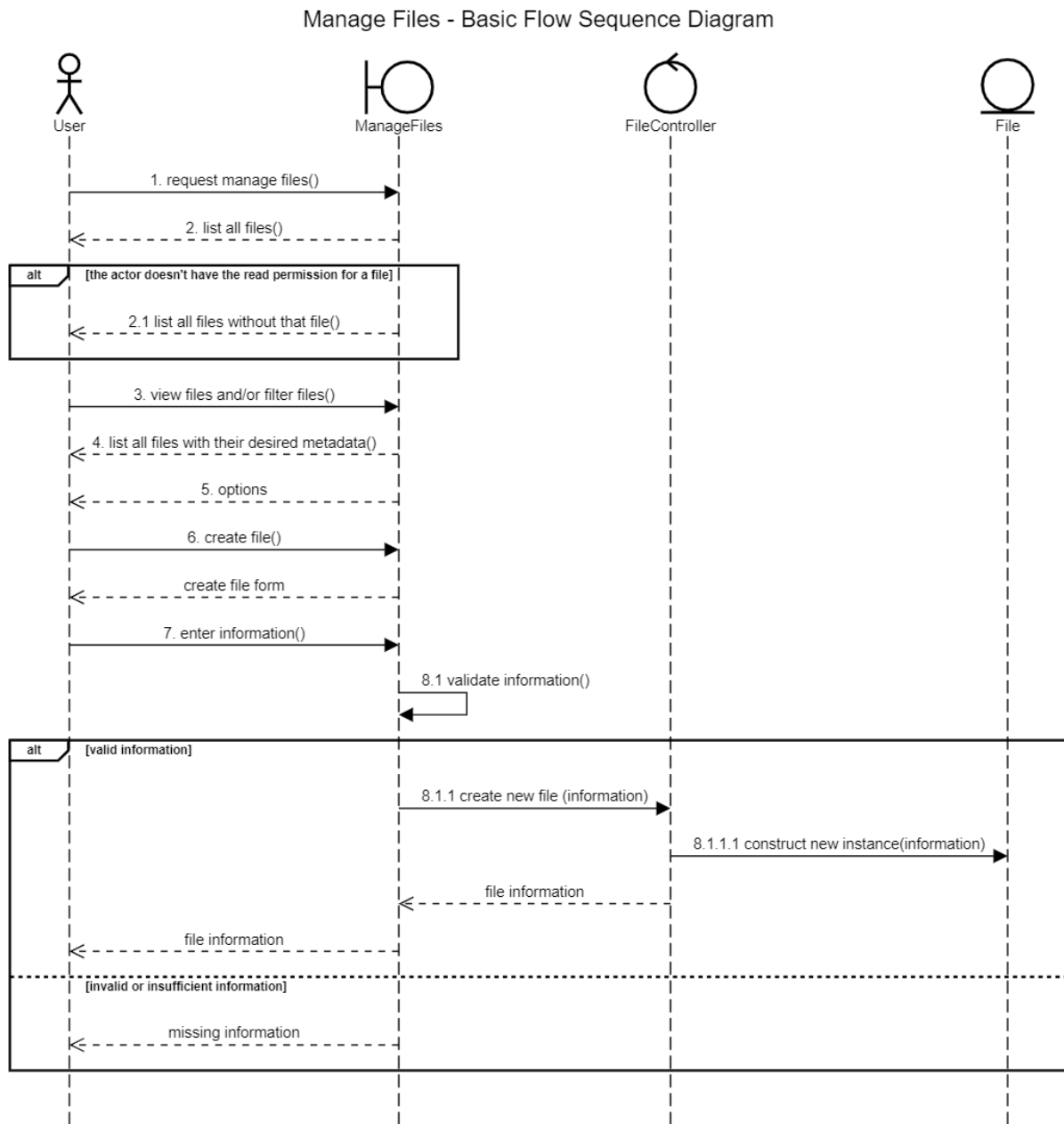


Figure 4.17: Manage Files - Basic Flow Sequence diagram

Update File - Sub flow

If the User wishes to edit a file by select the target file, the system will check if the User have the right to write the file. If the User does not have a write permission to a file, the system will displays en error message. Else, the system will display an update file f5.4 in Appendices). After the User enters the updated information, the system will check if the file exists or not. If the file exists, the system will save the updated information of the file. If not, the system displays an error message.

Manage Files - Update File Sub Flow Sequence Diagram

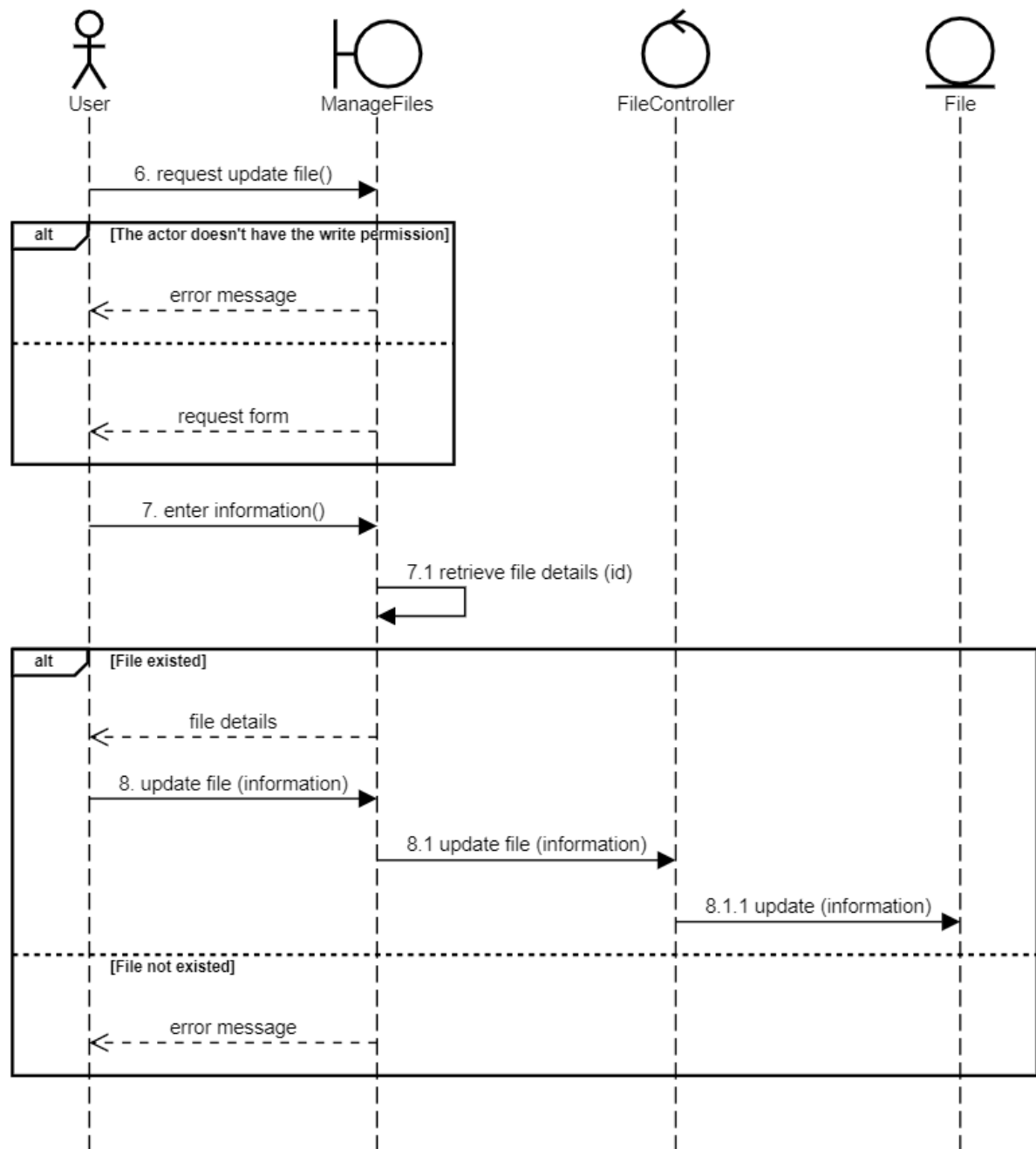


Figure 4.18: Manage Files - Update File Sub Flow Sequence diagram

Delete File - Sub flow

If the User wished to delete a file by selecting the target file, the system will check if the User have the right to write the file. If the User does not have a write permission to a file, the system would display an error message. Else, the system would display the full details

of the file. When User clicks on the "Delete" button, the system will check if the file exists or not. If the file exists, the system will remove the file. If not, the system displays an error message.

Manage Files - Delete File Sub Flow Sequence Diagram

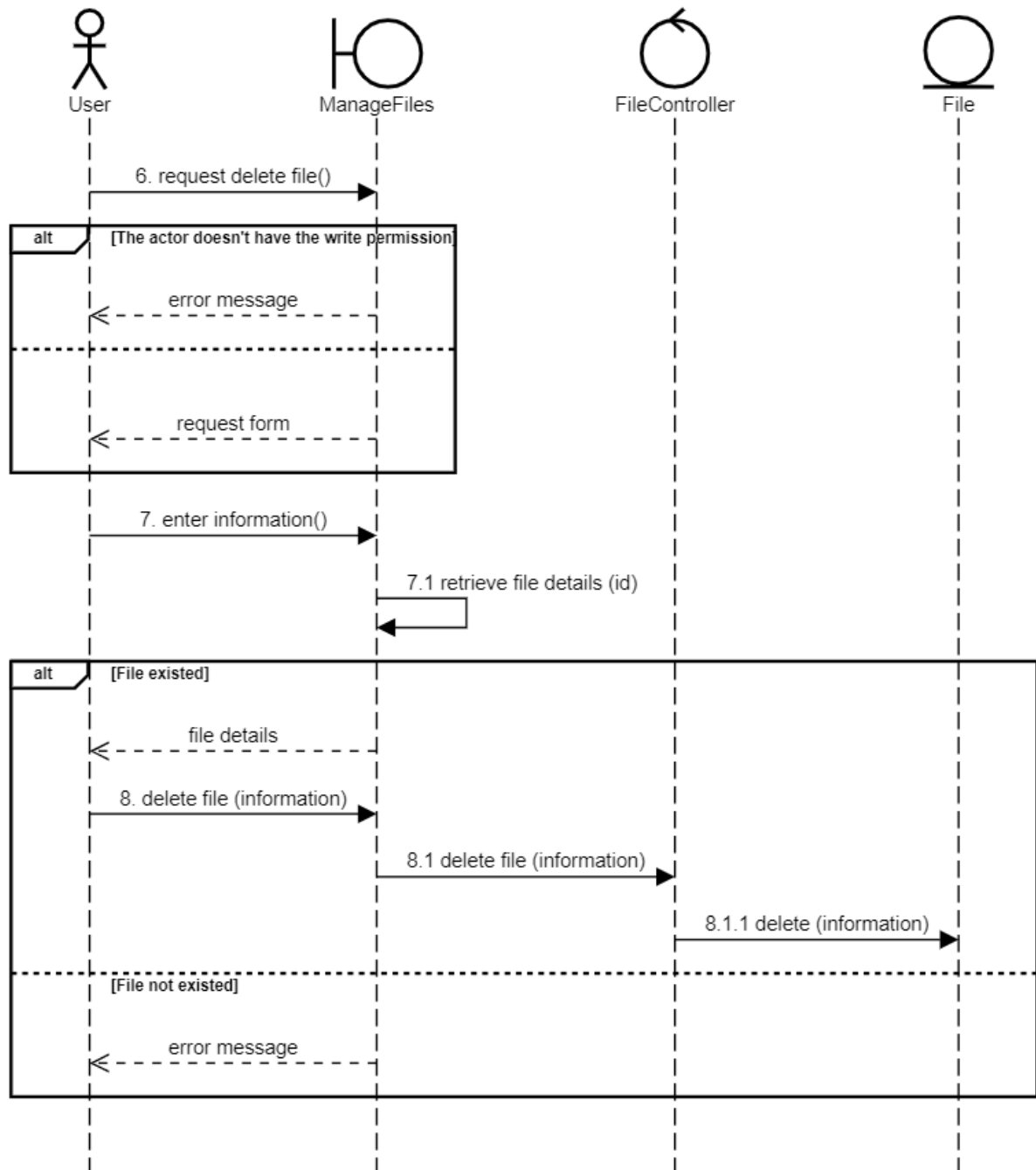


Figure 4.19: Manage Files - Delete File Sub Flow Sequence diagram

Move a File a Folder - Sub flow

If the User wishes to move a file, the system will check if the User have the right to write the file. If the User does not have a write permission to a file, the system will display an error message. Else, the system will display a move file form (Figure 13). The User will select a file as source and enter the targeted folder and the destination folder. The system will move the targeted file accordingly.

Manage Files - Move a File to a Folder Sub Flow Sequence Diagram

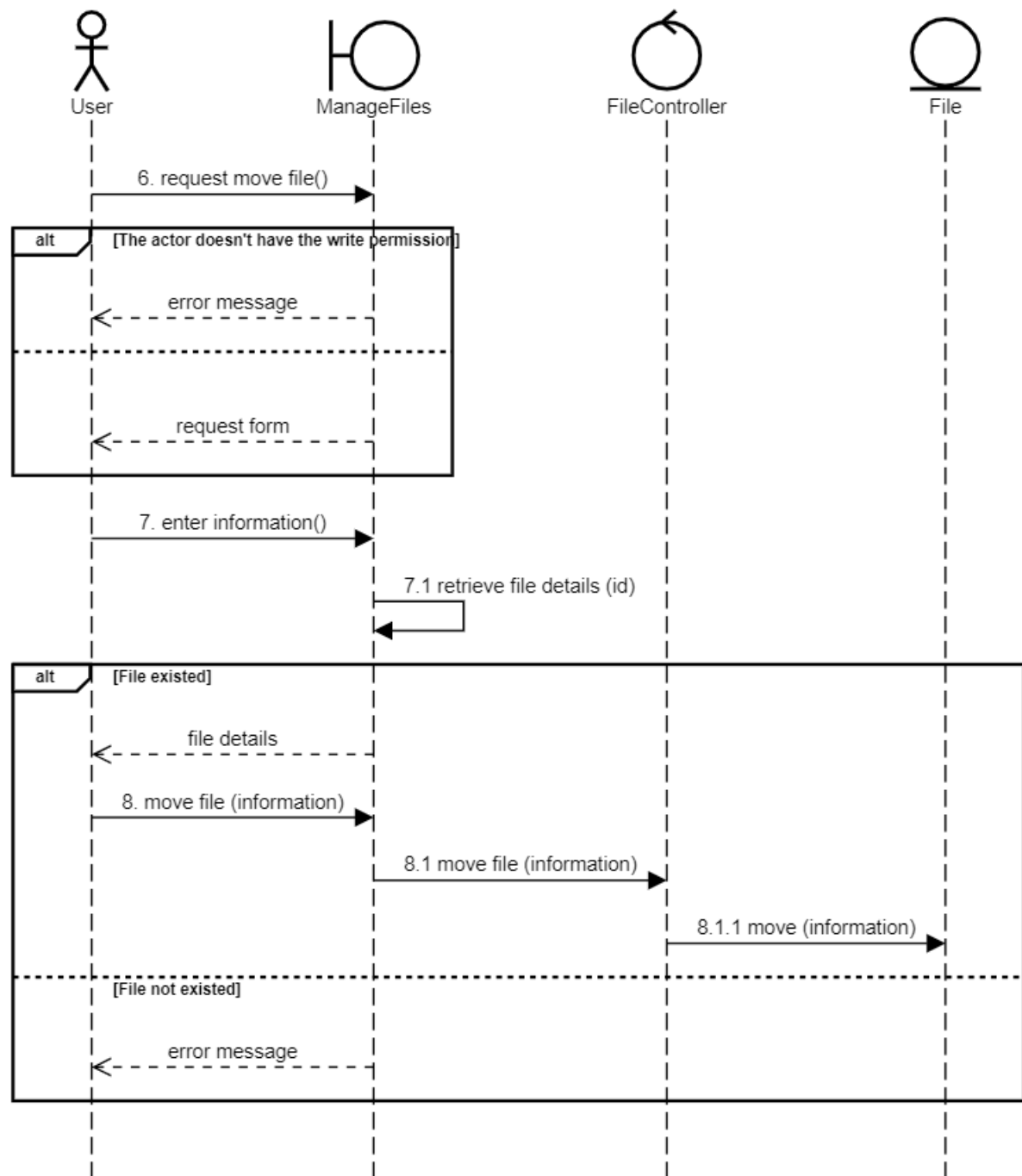


Figure 4.20: Manage Files - Move a File to a Folder Sub Flow Sequence diagram

Download File - Sub flow

If the User wishes to download a file, the system will check if the User have the right to view the file. If the User does not have a view permission to a file, the system will display an error message. Else, the system will process to return the file data for the user.

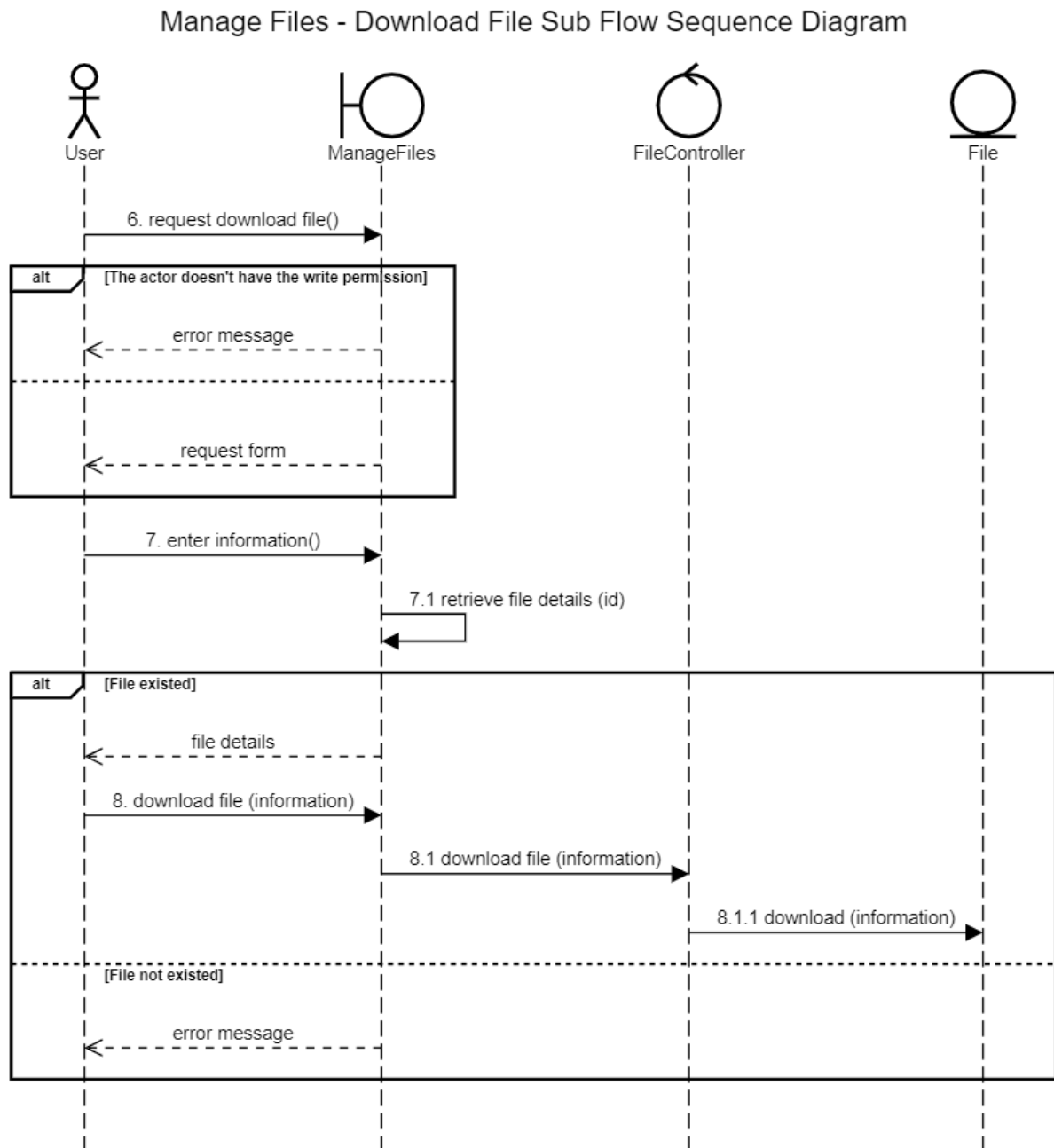


Figure 4.21: Manage Files - Download File Sub Flow Sequence diagram

4.3.7 Manage ACLs

This Use Case describes how a User can manage ACLs permission.

Manage ACLs - Basic flow

In the selected file or folder, the User wishes to manage ACL permissions (grant or delete). If the User chooses to grant, there will be a grant ACL form (Figure 5.3 and Figure 11 in Appendices). After the User enters all the required information, the system saves the ACL information, assigns a unique id to the ACL. Otherwise, the system displays an error message and asks the User to re-enter the information.

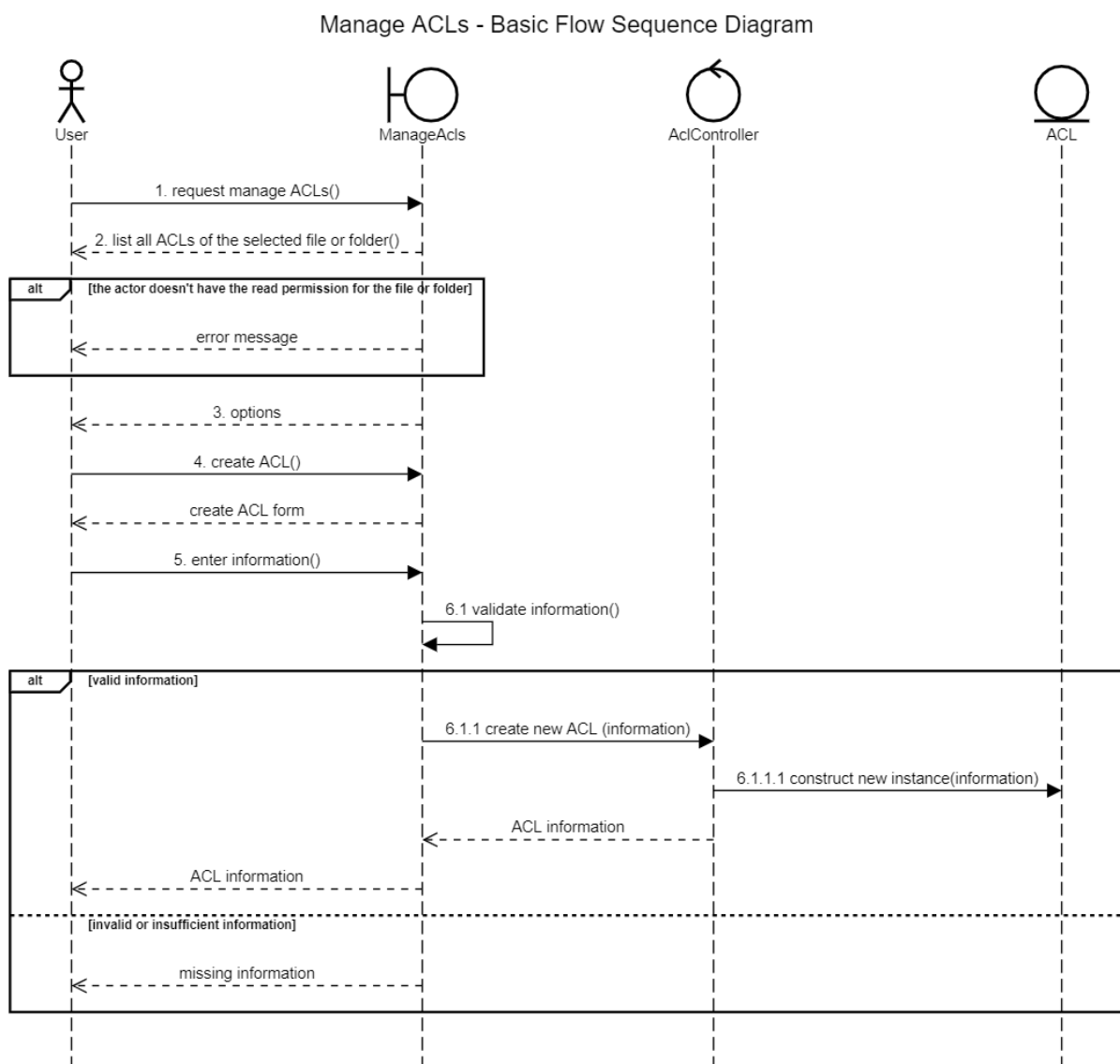


Figure 4.22: Manage ACLs - Basic Flow Sequence diagram

Delete ACL - Sub flow

If the User wished to delete a ACL by selecting the target ACL, the system will check if the User have the right to write the file or folder. If the User does not have a write permission, the system would display an error message. Else, the system would display the full details of the file or folder. When User clicks on the "Delete" button, the system will checks if the ACL exists or not. If the ACL exists, the system will remove the ACL. If not, the system displays an error message.

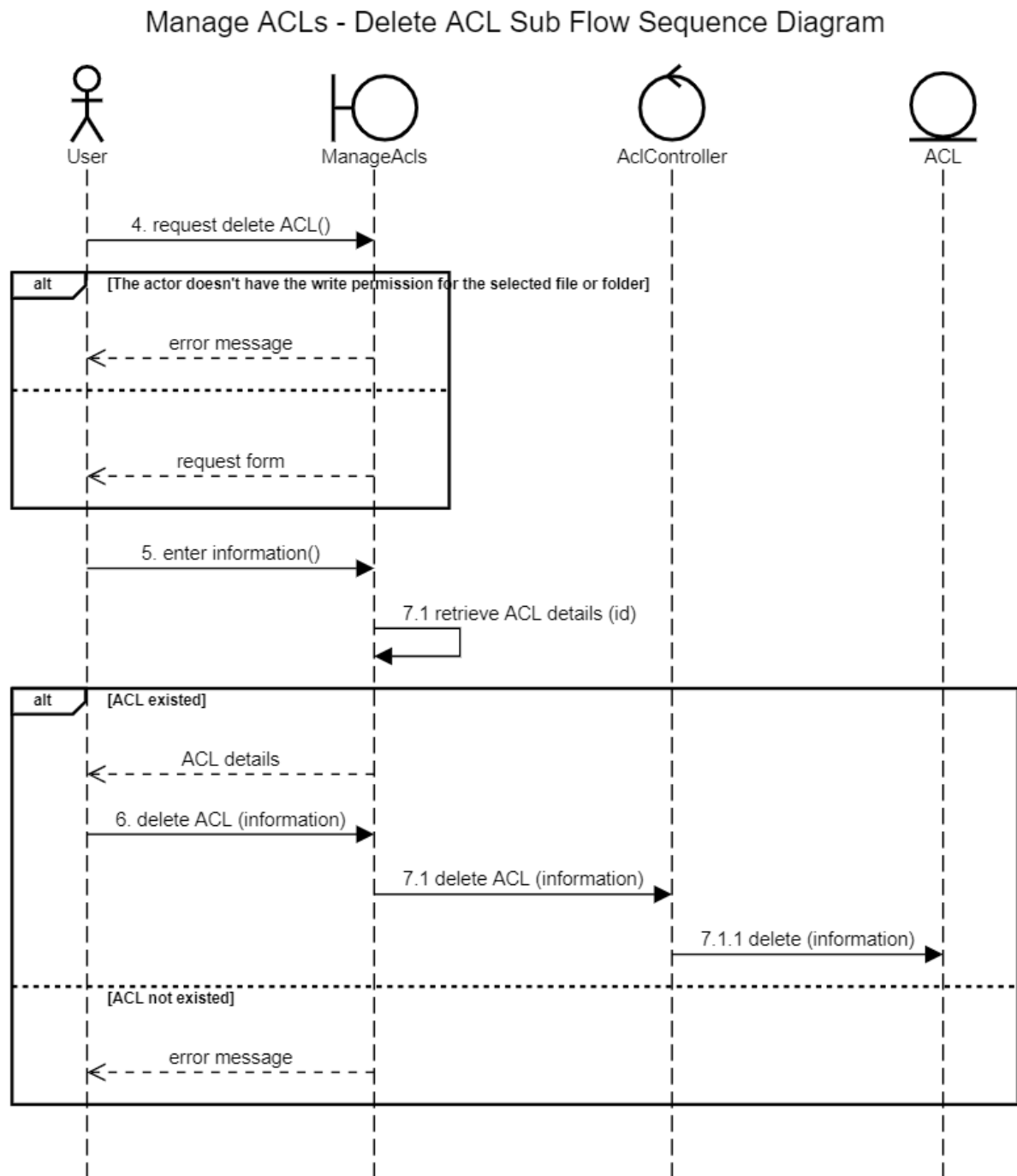


Figure 4.23: Manage ACLs - Delete ACL Sub Flow Sequence diagram

4.4 Tools and Techniques

4.4.1 SQL

With so much data and business relationships in our system, selecting a suitable database to manage it is critical. SQL is the clear choice because the project needs a close consistency between the data with high transactions and data security. While NoSQL has unstable schemas and is not fit for complex queries. Therefore, the best solution for our system is using SQL Database.[2]

4.4.2 Spring Boot

Spring Boot is based on the Spring framework and has several dependencies that can be integrated into a Spring application. Spring Kafka, Spring LDAP, Spring Web Services, and Spring Security are some examples. [3]

Spring Boot uses Java, which is one of the most popular programming languages in the world. It reduces development time, increases the development team's overall productivity, and helps autoconfigure all components for a production-grade Spring application.

The dependency injection element of the Spring framework focuses on offering flexibility. It facilitates the injection of required dependencies and the development of a loosely linked application. It is a lightweight framework that is compatible with many middleware services.

4.4.3 ReactJS

React.js is an open-source JavaScript package to create single-page apps' user interfaces. For web and mobile apps, it is used to manage the view layer. We can also make reusable UI components with React. [4]

React allows developers to create large web applications that can change data without reloading the page. The primary purpose of React is to be fast, scalable, and straightforward. It works only on user interfaces in the application. This corresponds to the view in the MVC template. It can be used with other JavaScript libraries or frameworks, such as Angular JS in MVC.

There are so many open-source platforms for making front-end web application development more accessible, like Angular. React has some benefits over other competitive technologies or frameworks. The component-based approach, well-defined lifecycle, and plain JavaScript make React very simple to learn, build a professional web (and mobile applications), and support it. React uses a special syntax called JSX, which allows us to mix HTML with JavaScript. It has a better learning curve than Angular because Angular is a 'Domain-specific Language'. React only requires basic knowledge of CSS and HTML.

For this project, according to the condition and the requirement of the system environment, React is a great choice compared to others.

4.4.4 Axios

Axios is a popular, promise-based HTTP client [5] that sports an easy-to-use API and can be used in both the browser and Node.js.

One of the most common activities a client-side JavaScript application will have to perform is making HTTP requests to fetch or save data. Third-party libraries have long been a popular approach to deal with more verbose browser APIs while abstracting away any variations across browsers.

Axios can be used in both the browser and Node.js compared with Fetch API built in many modern browsers. One such difference is in how the two libraries treat HTTP error codes [6]. When using Fetch, if the server returns a 4xx or 5xx series error, the `catch()` callback will not be triggered, and it is down to the developer to check the response status code to determine if the request was successful. On the other hand, Axios will reject the request promise if one of these status codes is returned.

One distinction that may prove to be a deal-breaker for some is status updates on uploads and downloads. We may register callback functions for `onUploadProgress` and `onDownloadProgress` to display the percentage completion in the app's UI because Axios is built on top of the older XHR API. Fetch cannot currently do so. Therefore, the best choice is using Axios.

4.4.5 Material-UI

Material-UI is based on Google's Material Design guidelines. Like other design systems, Material Design was intended to provide a uniform user experience across various devices, platforms, and input methods. Google adopted Material Design to ensure that users had a uniform user experience regardless of how they accessed their products. Apple embraced flat design principles as their standard. [7]

Material-UI is the world's most popular React UI framework. It provides a strong foundation for building dynamic UIs. It is blazing fast. It uses JSS at its core – a high performance JavaScript to CSS compiler which works at runtime and server-side, with less than 15 KB gzipped; and no bundle size increase if used alongside Material-UI.

4.4.6 Swagger

API Documentation is needed because the application is connected to many other services. It will improve user adaption, save support time and costs, and easier maintenance.

Swagger with API description formats like the OpenAPI/Swagger Specification is an excellent choice for that purpose. It has automated the documentation process, making it easier for teams to generate and maintain them. [8]

4.4.7 JWT (JSON Web Token)

JWT is now widely used for authentication and data exchange. The Server encodes data into a JSON Web Token and transmits it to the Client instead of starting a Session

(Session-based Authentication). After the Client saves the JWT, it should be appended to every request from the Client to protected routes or resources (commonly at header). The Server will validate the JWT, and the response will be returned.

Compared to Session-based Authentication, which requires the storage of the session on a cookie, JWT (Token-based Authentication) has the substantial advantage of storing the token on the client-side: Local Storage for browsers, Keychain for iOS, and SharedPreferences for Android... As a result, we will not need to create a separate backend project for Native Apps or a separate Authentication module for Native App users.

For that, using JWT is ideal for this project.

4.4.8 RBAC and ACL Authorization

We developed the authentication from JWT, which establishes the principal's identity. After then, authorization is required, which determines what the principal is permitted to do. We start with role-based authorization (RBAC), which involves assigning roles to specific users and then utilizing those roles to define what they may see and do, ideally through related permissions. Role Admin and Role User are two types of roles.

When we have to maintain predicates on principals regarding specific targets, the problem occurs. When we discuss the data lake, we want to make it so that only the file owner and the admin may update a file after it has been posted and that no one else can do so without authorization. "Can User1 edit files in general?" is a question that role-based permission helps us answer. ACL-based authorization (Access Control List) is more fine-grained, requiring the management of relationships between actors and targets to answer the question, "Can User1 edit file 10?". [9]

We implement ACL Authorization Model with the help of Spring Security's ACL module.

Chapter 5

Results and Discussion

5.1 Result

After three months of development, the APIs and Dashboard for Data Lake has basically been completed, meeting the initial criteria. To be more specific, a website has been designed with the following functions:

- **Register and Login:** Allows users to create an account and log in using the registered account based on username and a real email.
- **Manage Users:** Allow Admin to read, create, edit, and delete information about users.
- **Manage User Groups:** Allow Admin to read, create, edit, and delete information about groups.
- **Manage Folders and Manage Files:** Authorized users are allowed to read, create, edit, and delete information about files and folders. To increase productivity, users can filter them by custom criteria.
- **Manage ACLs:** Authorized users are allowed to grant permission for other users on the files or folders they selected.

5.1.1 API

These are all the endpoints for the API we have been building:

Table 5.1: All API Endpoints

Methods	URLs	Action
POST	/api/auth/signup	Signup new account
POST	/api/auth/signin	Login an account

POST	/api/auth/refresh-token	Get access token from re-fresh token
GET	/api/auth/logout	Logout current user
GET	/api/users/check_email	Check if Email is available to use
GET	/api/users/my-profile	Get the logged in's user profile
GET	/api/users/{id}	Get user by ID
PUT	/api/users/{id}	Update user by ID
DELETE	/api/users/{id}	Delete user by ID
GET	/api/users/check_username	Check if Username is available to use
GET	/api/users/find/username	Get user by user name
GET	/api/users	Get a list of all users in the system
GET	/api/groups/{id}	Get a group by ID
DELETE	/api/groups/{id}	Delete a group
PUT	/api/groups/{groupId}/users/{username}	Add a user to a group
GET	/api/groups/find/{name}	Get a group by name
GET	/api/groups	Get all groups
POST	/api/groups	Add an user group
GET	/api/find/name/{name}	Find all files and folders by name containing
GET	/api/find/topics	Get files and folders by topics
GET	/api/folders/ls/{folderId}	List all files and subfolders inside the folder by ID
GET	/api/content	Get all first node files and folders
GET	/api/folders	Get all folders
POST	/api/folders	Add a folder
GET	/api/folders/{id}	Get a folder by ID
PUT	/api/folders/{id}	Update a folder by ID
DELETE	/api/folders/{id}	Delete a folder
GET	/api/folders/content/{id}	Get a list of content inside folder by ID
PUT	/api/folders/{folderId}/subfolders/{subfolderId}	Add a subfolder to folder
PUT	/api/folders/{folderId}/files/{fileId}	Add a file to a folder
GET	/api/files/data/{id}	Get File Data

GET	/api/files/{id}	Get a file by ID
PUT	/api/files/{id}	Update a file by ID
DELETE	/api/files/{id}	Delete a file by ID
GET	/api/folder/{folderId}/files	Get All Files by Folder Id
GET	/api/files	Get all files
POST	/api/files	Upload a file
PUT	/api/acl/file/user	Grant File Permission For User
DELETE	/api/acl/file/user	Remove a File Permission For User
GET	/api/acl/perm	Get All Permissions
PUT	/api/acl/folder/user	Grant Folder Permission For User
DELETE	/api/acl/folder/user	Remove a File Permission For User
PUT	/api/acl/file/group	Grant File Permission For Group
DELETE	/api/acl/file/group	Remove a File Permission For Group
PUT	/api/acl/folder/group	Grant Folder Permission For Group
DELETE	/api/acl/folder/group	Remove a Folder Permission For Group

5.1.2 Web Dashboard

Following are the screenshots for the Manage Files Use Case, of the Web Dashboard we have been building. The rest is included in the Appendices section.

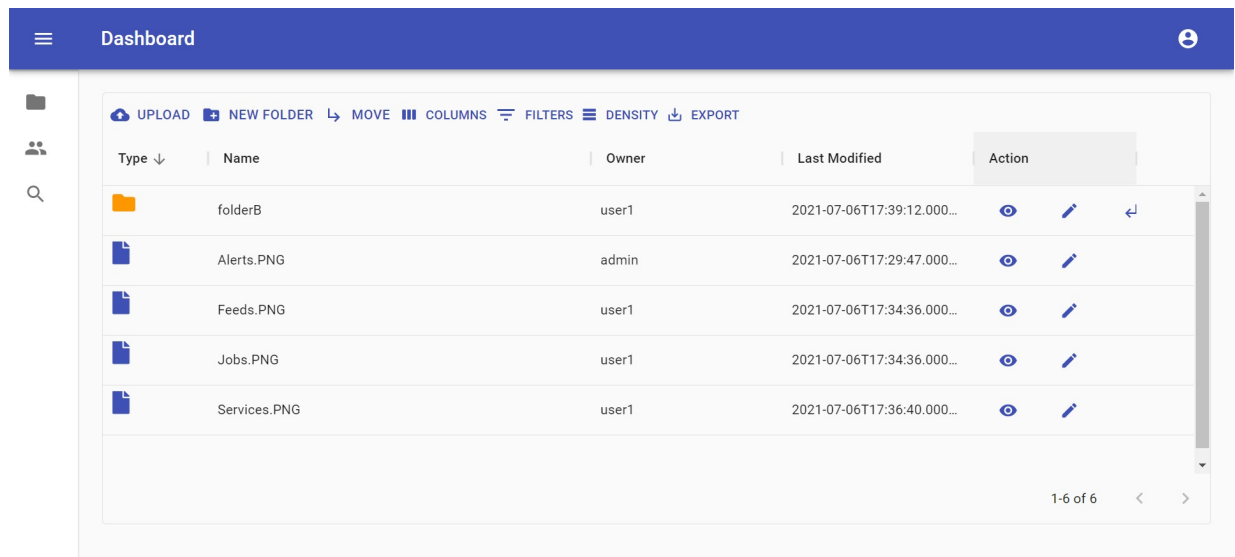


Figure 5.1: List all files and folders user interface

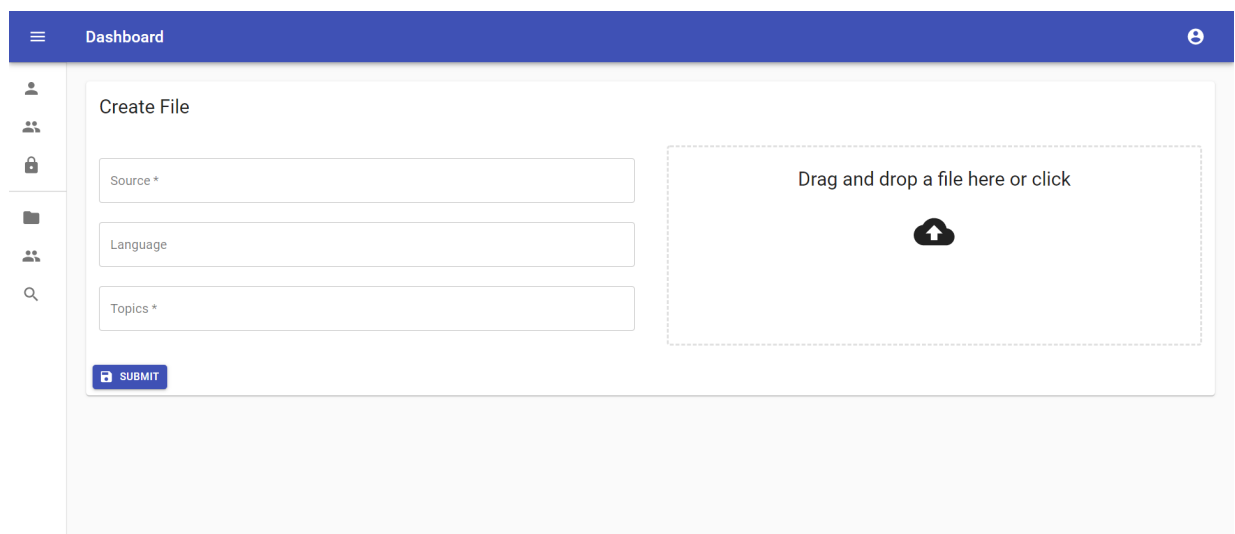


Figure 5.2: Upload File user interface

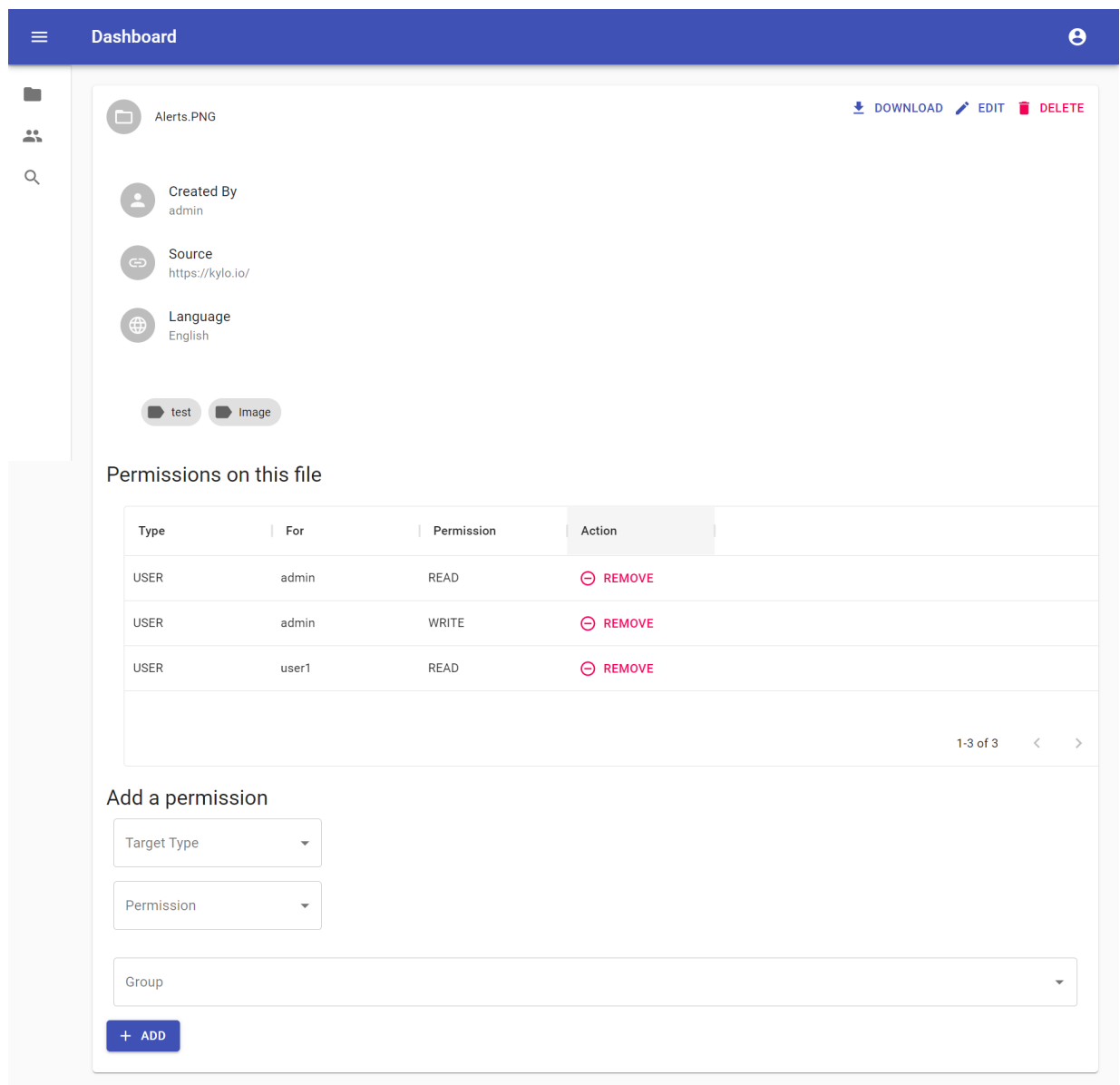


Figure 5.3: View File Details user interface

The screenshot shows a web application interface for editing a file. The top navigation bar is blue and contains a menu icon, the word "Dashboard", and a user profile icon. A left sidebar is visible with icons for a folder, users, and search. The main content area is titled "Edit File #1" and contains the following elements:

- A "Name" input field with the value "Alerts.PNG".
- A "Source *" input field with the value "https://kylo.io/".
- A "Language" input field with the value "English".
- Two radio buttons labeled "test" and "Image".
- A "Topics *" text area.
- Two buttons at the bottom: "SAVE" and "DELETE".

Figure 5.4: Edit File user interface

5.2 Discussion

Despite being implemented completely, the main functions in this project have some existing problems.

The concept of Data Lake is new to us. And it takes a long time to get a sense of what it is and how to do.

Furthermore, because this website was not created by a team of experienced creators but by a novice individual, it contains numerous flaws in terms of design and features.

Finally, time constraints prevent the testing process from being completed completely and methodically, potentially resulting in a large number of minor problems. On the other hand, this website has accomplished all of the initial objectives listed in the Objectives chapter.

Chapter 6

Conclusions

6.1 Conclusion

In conclusion, the purposes and objectives of the APIs and Dashboard for Data Lake are achieved. By providing a user-friendly interface on the web, managing data becomes more accessible and more effective.

The main functions have already been implemented:

- User can register and log in the system.
- Admin can manage users and user groups.
- User can upload, update and remove any files and folders.
- User can grant or remove permission to file or folders they own.

6.2 Future Works

To further improve this web application in the future, the following tasks need to be done:

- Improve web security.
- Forgot password.
- Account verification by email.
- Scale better when upload and download files.
- Upload Folder.
- Create a Guideline for using the system.
- Improve the performance of the application to be more responsive and user-friendly.

Glossary

Alternative flow Actions that are not compulsory for the Use Case but they are necessary to get all provided function of system..

Application package A collection of programs or modules that is directed at some generic application and can be tailored (perhaps with some additions) to the needs of a specific instance of that application..

Basic flow The set of required actions to perform Use Case in the system..

Boundary class a class that is the boundary of the system and other system or user (which is actor in the use case diagram)..

Brief Description The models of discourse along with exposition, argumentation and narration about summary, meaning, role of Use Cases..

Control class A control class manages the flow of interaction of the scenario. A control class to achieves use cases in the use case diagram..

Entity class An entity is a long-lived, passive class that is responsible for some meaningful chunk of information. that class has data..

Flow of Events The set of actions ordered to perform a Use Case in the system..

Post-Condition a statement or statements describing the condition that will be true when the operation has completed its task. If the operation is correct and the pre-condition(s) met, then the post-condition is guaranteed to be true..

Pre-Condition a statement or set of statements that outline a condition that should be true, or conditions that should be true, when the operation is called. The operation is not guaranteed to perform as it should unless the pre-conditions have been met..

Sequence diagram an interaction diagram that shows how processes operate with one another and in what order, and shows object interactions arrange in time sequence..

Acronyms

ACL Access Control List.

API application programming interface.

DB database.

DBMS database management system.

HTTP Hypertext Transfer Protocol.

JDBC Java Database Connectivity.

JSON JavaScript Object Notation.

RBAC Role-Based Access Control.

SQL structured query language.

USTH University of Science and Technology of Hanoi.

VAST Vietnam Academy of Science and Technology.

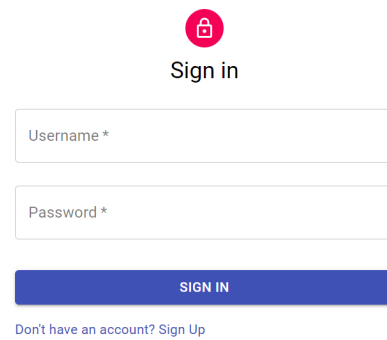
Bibliography

- [1] T. John and P. Misra, *Data Lake for Enterprises: Lambda Architecture for Building Enterprise Data Systems*. Packt Publishing, 2017.
- [2] L. Ullman, *MySQL, Second Edition (Visual QuickStart Guide)*. USA: Peachpit Press, 2006.
- [3] R. Johnson, J. Hoeller, A. Arendsen, T. Risberg, and D. Kopylenko, *Professional Java Development with the Spring Framework*. GBR: Wrox Press Ltd., 2005.
- [4] A. Fedosejev, *React.Js Essentials*. Packt Publishing, 2016.
- [5] “Official axios npm documentation.” [Online]. Available: <https://www.npmjs.com/package/axios>
- [6] M. S. David Gourley, Brian Totty, *HTTP: The Definitive Guide: The Definitive Guide (Definitive Guides)*. O’Reilly Media, 2002.
- [7] V. K. Kotaru, *Material Design Implementation with AngularJS: UI Component Framework*, 1st ed. USA: Apress, 2016.
- [8] I. Koren and R. Klamma, “The exploitation of openapi documentation for the generation of web frontends,” in *Companion Proceedings of the The Web Conference 2018*, ser. WWW ’18. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2018, p. 781–787. [Online]. Available: <https://doi.org/10.1145/3184558.3188740>
- [9] W. Wheeler and J. White, *Spring in Practice*, 1st ed. USA: Manning Publications Co., 2013.

Appendices

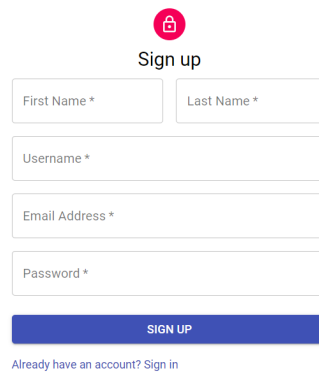
List of Figures in Appendices

1	Login user interface	82
2	Register user interface	82
3	Admin List All Users user interface	83
4	Admin Create User user interface	83
5	Admin Edit and Delete a User user interface	84
6	Admin List all Groups user interface	84
7	User Limited View List all Groups user interface	85
8	Admin Create Group user interface	85
9	Admin Update Group user interface	86
10	Create Folder user interface	86
11	View Folder Details user interface	87
12	Edit Folder user interface	88
13	Move File or Folder user interface	88
14	Update Profile user interface	89
15	Search ACL sources user interface	89
16	Admin List all ACLs user interface	90
17	Admin Grant a ACL user interface	90



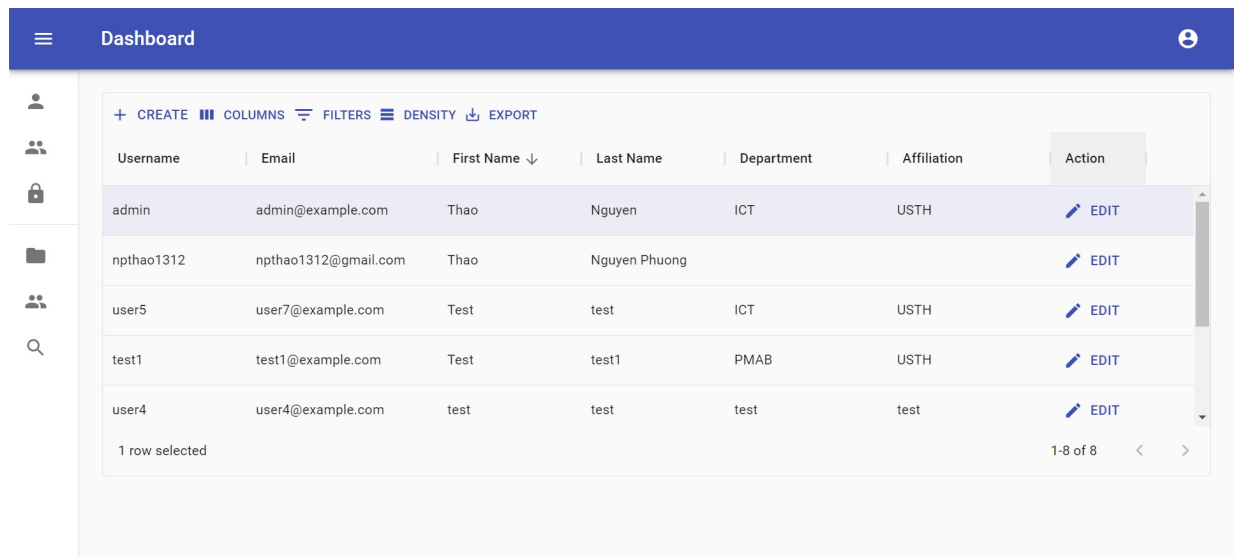
The sign in interface features a red circular icon with a white padlock at the top center. Below it is the text "Sign in". The form consists of two white input fields: "Username *" and "Password *". Below these fields is a blue button with the text "SIGN IN" in white. At the bottom, there is a link that says "Don't have an account? Sign Up".

Figure 1: Login user interface



The sign up interface features a red circular icon with a white padlock at the top center. Below it is the text "Sign up". The form consists of five white input fields: "First Name *", "Last Name *", "Username *", "Email Address *", and "Password *". Below these fields is a blue button with the text "SIGN UP" in white. At the bottom, there is a link that says "Already have an account? Sign in".

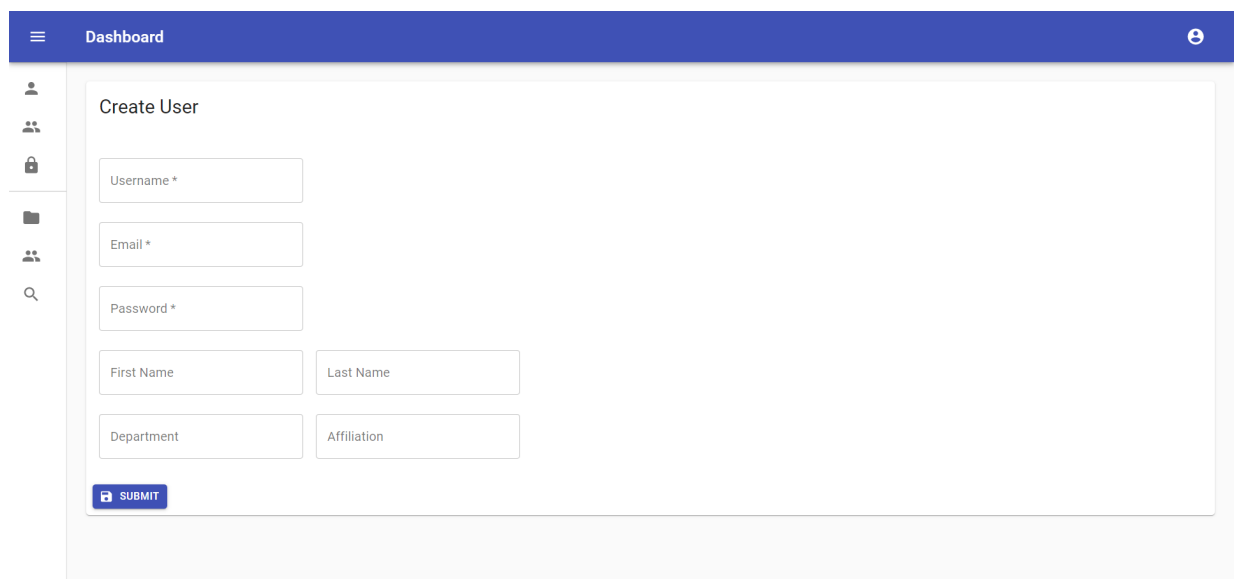
Figure 2: Register user interface



Username	Email	First Name	Last Name	Department	Affiliation	Action
admin	admin@example.com	Thao	Nguyen	ICT	USTH	EDIT
npthao1312	npthao1312@gmail.com	Thao	Nguyen Phuong			EDIT
user5	user7@example.com	Test	test	ICT	USTH	EDIT
test1	test1@example.com	Test	test1	PMAB	USTH	EDIT
user4	user4@example.com	test	test	test	test	EDIT

1 row selected 1-8 of 8

Figure 3: Admin List All Users user interface



Create User

Username *

Email *

Password *

First Name Last Name

Department Affiliation

[SUBMIT](#)

Figure 4: Admin Create User user interface

Edit User #9

test1

test1@example.com

First Name
Test

Last Name
test1

Department
PMAB

Affiliation
USTH

SAVE **DELETE**

Figure 5: Admin Edit and Delete a User user interface

Dashboard

+ CREATE || COLUMNS FILTERS DENSITY EXPORT

Name	Users	Actions
demo	user1	+ ADD DELETE
test	user1, user5, user2	+ ADD DELETE

1-2 of 2 < >

Figure 6: Admin List all Groups user interface

Name	Users
demo	user1
test	user2, user1, user5

Figure 7: User Limited View List all Groups user interface

Create Group

Name*

Figure 8: Admin Create Group user interface

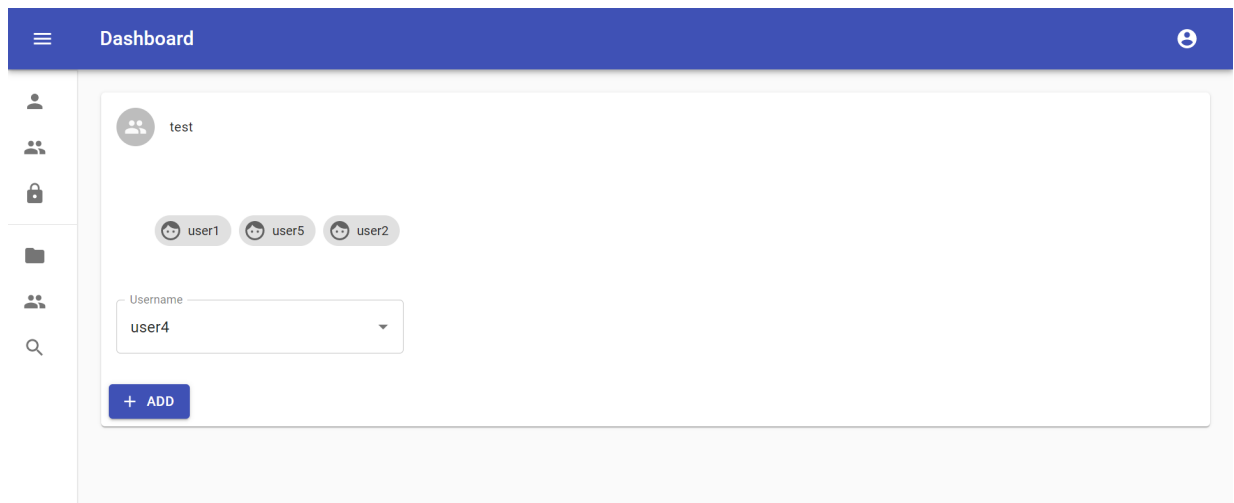


Figure 9: Admin Update Group user interface

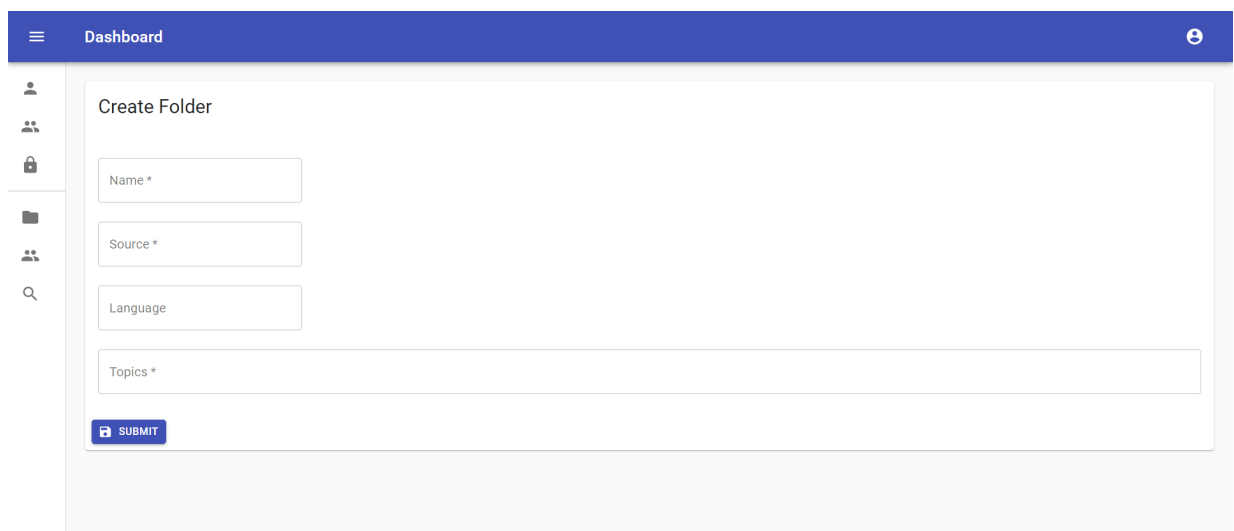


Figure 10: Create Folder user interface

Dashboard

folderB

EDIT

DELETE

Created By

user1

Source

https://github.com/ciur/papermerge/tree/master/papermerge/wsignals

Language

English

test

Education

Permissions on this folder

Type	For	Permission	Action
USER	user1	READ	REMOVE
USER	user1	WRITE	REMOVE

1-2 of 2

Add a permission

Target Type

Permission

Group

+ ADD

Figure 11: View Folder Details user interface

The screenshot shows a web application interface with a blue header bar labeled "Dashboard" and a user profile icon. On the left, there is a sidebar with icons for a folder and users. The main content area is titled "Edit Folder #2" and contains the following fields and elements:

- Name:** A text input field containing "folderB".
- Source *:** A text input field containing "https://github.com/ciur/paperm".
- Language:** A text input field containing "English".
- Tags:** Two toggle buttons labeled "test" and "Education".
- Topics *:** A large empty text input field.
- Actions:** Two buttons at the bottom: a blue "SAVE" button with a floppy disk icon and a red "DELETE" button with a trash icon.

Figure 12: Edit Folder user interface

The screenshot shows a web application interface with a blue header bar labeled "Dashboard" and a user profile icon. On the left, there is a sidebar with icons for a folder, users, and a search icon. The main content area is titled "Move" and contains the following fields and elements:

- File:** A dropdown menu currently showing "File".
- Move File or Folder?:** A label text.
- From File:** A dropdown menu showing "Tables.PNG".
- To Folder:** A dropdown menu showing "folderA".
- Action:** A blue "SUBMIT" button with a floppy disk icon.

Figure 13: Move File or Folder user interface

Edit Profile

admin

admin@example.com

First Name
Thao

Last Name
Nguyen

Department
ICT

Affiliation
USTH

UPDATE PROFILE

Figure 14: Update Profile user interface

Search Topics *

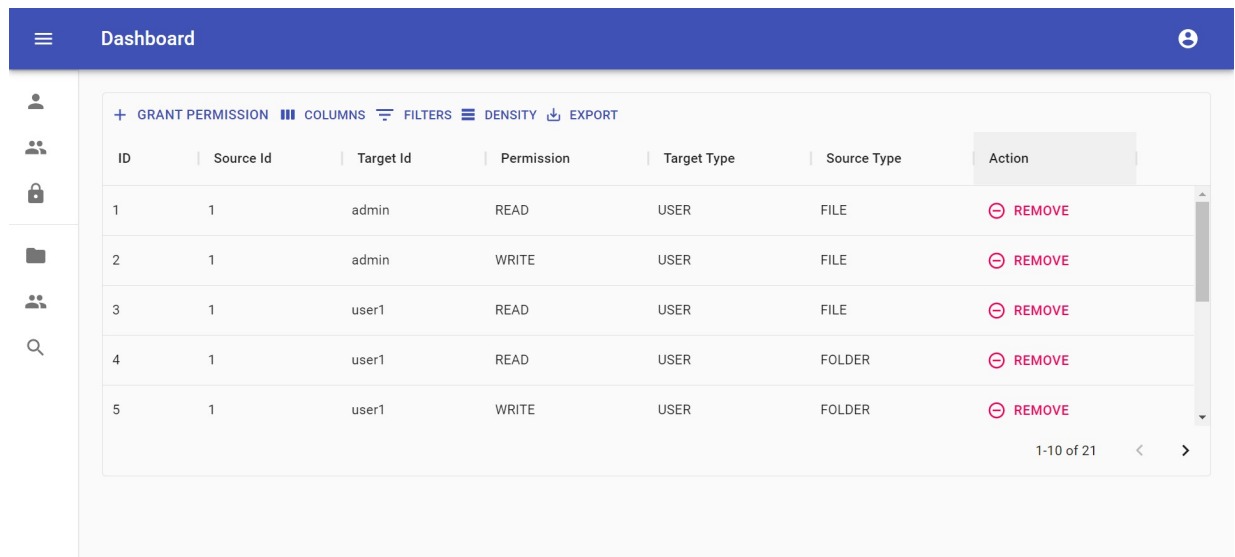
test Image Search Topics

COLUMNS FILTERS DENSITY EXPORT

ID	Name	Source	Language	Type	Topics
1	folderA	test		inode/directory	test,Image Classification
2	folderB	test		inode/directory	test,pandas
3	subfolder	test		inode/directory	test,Wine
4	profile.png	unspecified	English	image/png	test,image
5	token.json	unspecified	English	application/json	test,json

1-10 of 37

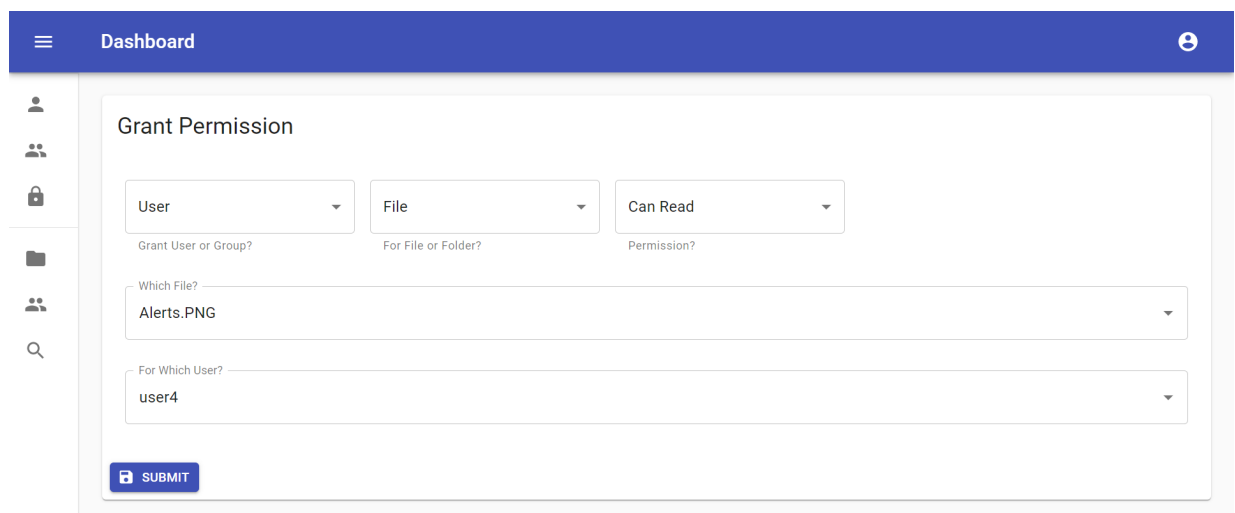
Figure 15: Search ACL sources user interface



ID	Source Id	Target Id	Permission	Target Type	Source Type	Action
1	1	admin	READ	USER	FILE	⊖ REMOVE
2	1	admin	WRITE	USER	FILE	⊖ REMOVE
3	1	user1	READ	USER	FILE	⊖ REMOVE
4	1	user1	READ	USER	FOLDER	⊖ REMOVE
5	1	user1	WRITE	USER	FOLDER	⊖ REMOVE

1-10 of 21

Figure 16: Admin List all ACLs user interface



Grant Permission

User Grant User or Group? File For File or Folder? Can Read Permission?

Which File? Alerts.PNG

For Which User? user4

SUBMIT

Figure 17: Admin Grant a ACL user interface