System Software Cloud File Storage

N0992216

April 2023

1 Abstract

1.1 Background

Cloud storage is a big part of many people's lives. Users can now easily save a file to a cloud storage server and access it on any device. There are a wide number of reliable cloud storage providers, for example, Dropbox and Google Drive. This has allowed for easier access to files and secure storage of the files. In this coursework, we were given the task to create a cloud file storage.

1.2 Results

Through this coursework, my team was able to create a viable cloud storage system. My team used Java, Docker and SQLite to achieve this. We allowed users to register their details to the program and sign in. Once the user was logged in they could do file management which includes creating files, deleting files etc. As well as this they could enter a terminal emulator. This acts like a terminal within the GUI, this allows users to use commands like ls, ps and nano etc.

1.3 Conclusion

In the end, we created a viable product. My team developed a better understanding of Docker, Doxygen, Java and SQLite.

2 Introduction

Imagine after a hard day of work saving your unfinished work onto a USB drive. The next day you come back in, ready to finish the work. When you plug in your drive to see an Error message. How would you feel? Angry? Annoyed? This was the reality for many people before Cloud Storage. Cloud Storage allows users to safely and securely store their files. They can then open said files on any compatible device free of charge (to an extent). Cloud Storage has become a staple in many lives, including mine, being part of many business processes.

In this project, my team member and I will be attempting to create a cloud file storage using Docker, Java and SQLite.

3 Background Research

To create the application we used the Netbeans IDE. We chose this as it is a flexible, simple IDE. This allows us to find bugs faster. We used JavaFX scene builder and Java for the back end. JavaFX scene builder is a drag-and-drop interface. This allows us to create the front end faster. Since JavaFX derives from Java, we used Java to create the back end. Furthermore, Java has efficient memory management so the program will run faster, this makes it superior to many other languages. We used SQLite to create the database as it is a lightweight database. We used Docker to assist us in making the cloud storage system. Finally, we used Doxygen to create a reference manual for the project.

4 Design

4.1 Assumptions

From first impressions of the project, we thought multiple tables were used for the project. One for the files and one for the users. However, we found a more efficient way of doing this.

4.2 Coding standards

We will be following the Java coding standards because it makes the code easier to read. As a result, it will be more maintainable. This will allow me or my partner to understand changes made by the other member.

I will be providing examples of some of the coding standards we tried to follow but for more standards please visit here https://www.geeksforgeeks.org/coding-guidelines-in-java/. Firstly, we will be using curly braces. We will be using it to define classes, functions, loops etc. This will make it clear to see the classes and functions. Furthermore, we will be using Indentation. This will be used to separate alike variables, loops etc. Moreover, helps with adding spaces between keywords. Finally, we will be using the comments. This will be mainly used in conjunction with Doxygen. It will enable the other team member to read a short description of how the code works.

5 Implementation

We stored the user's information using an SQLite database. Since passwords are sensitive information we decided to salt and hash the passwords before storing them. To make the Cloud Storage system I used docker. Through docker, I was able to create two containers, one for file storage and one for the application.

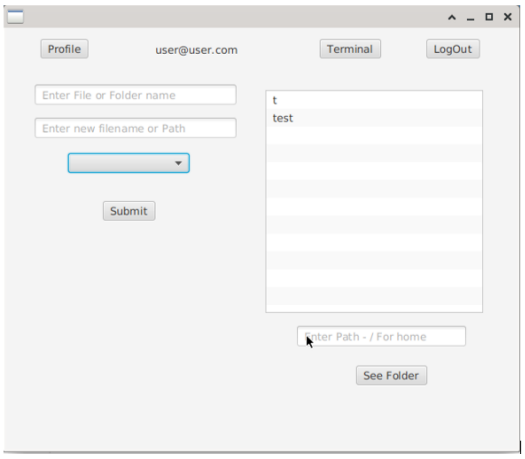

Figure 1: Main interface of Application

I then created a volume which created a file for the data. This volume would then be connected to both the application and file storage container. Hence when a user creates a file it is uploaded to the file storage container through the volume.

Figure 1 shows the main interface of the program. This page allows for users to view files and folders. As well as do all their file management using the combo box. If the commands are not used properly error messages will be displayed.

Figure 2 shows a general overview of the system. When the application first starts a user can log in or register. Once logged in they are sent to the file management screen which is the main interface. From here they can manage all their files. Next, they can either log out, go to the terminal or go to their Profile. Once the user has fully finished all their tasks they can log out.

Features I implemented:

User management - creating, deleting, updating, login and logout

Encryption of passwords

Errors and success messages displayed to users

File management - create new files, delete files, rename files, move files, download files.

File Storage - storing files in a separate docker container

Terminal emulation including all the commands.

Multiple users can access the file at the same time

User cannot be logged into multiple instances of the application

Files can be shared with other users with read-only or read-write access

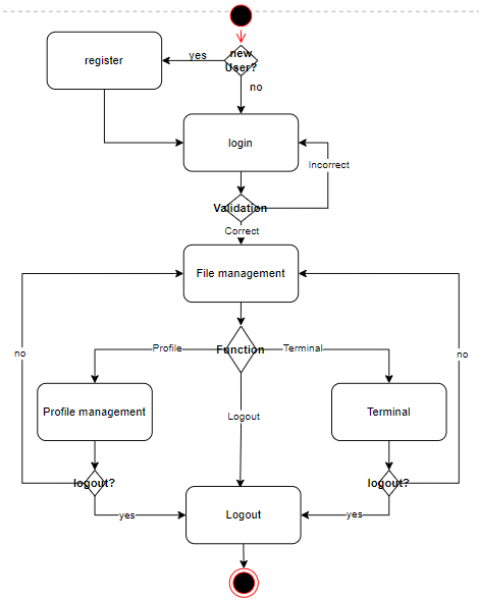

Figure 2: Activity Diagram for Whole System

6 Testing

6.1 Test Plan

1 - The user registers with a valid email and password - Success: User is registered - Correction procedure: show error message. This will most likely be a system error review system.

2- The user registers with an invalid email and password - Success: User is shown error message - Correction procedure: show error message and provide context of incorrect text field.

3 - User login with valid email and password - Success: User is logged in - Correction procedure: Output error message. This will most likely be a system error review system.

4 - User login with invalid email and password - Success: User is shown error message - Correction procedure: show error message and provide context of incorrect text field.

5 - The user tries to create a file - Success: File is created - Correction procedure: Output error message with instructions on how to use GUI.

6 - The user tries to delete a file - Success: File is deleted - Correction procedure: Output error message with instructions on how to use GUI.

7 - The user attempts to rename a file - Success: File is renamed - Correction procedure: Output error message with instructions on how to use GUI.

8 - The user attempts to move a file from one folder to another - Success:

File is moved to other folder - Correction procedure: Output error message with instructions on how to use GUI.

9 - The user attempts to run the mv command - Success: File is moved - Correction procedure: Display error message and show user correct format of command.

10 - The user attempts to run the ps command - Success: shows current processes - Correction procedure: Display error message and show user correct format of command.

11 - The user attempts to run the nano command - Success: opens nano - Correction procedure: Display error message and show user correct format of command.

12 - The user attempts to run the whoami command - Success: User email outputted - - Correction procedure: Display error message and show user correct format of command.

13 - The user attempts to run the ls command - Success: lists files in current folder - Correction procedure: Display error message and show user correct format of command.

14 - The user attempts to run the mkdir command - Success: makes new folder - Correction procedure: Display error message and show user correct format of command.

15 - The user attempts to run the cp command - Success: copies file - Correction procedure: Display error message and show user correct format of command.

16 - The user attempts to run the tree command - Success: Show tree for current directory - Correction procedure: Display error message and show user correct format of command.

17 - Run multiple instances of the application - Success: Multiple instances can be run - Correction procedure: Review application code

18 - Try to login into the same user to multiple instances of the application - Success: Denies user entry if already logged in - Correction procedure: check currentusers.txt to see if the email is correctly appended. If not review application code.

19 - The user shares a file with read-only access to another user - Success: Shares file in read-only mode - Correction procedure: Output error message with instructions on how to use GUI.

19 - The user shares a file with write-read access to another user - Success: Shares file in read-write mode - Correction procedure: Output error message with instructions on how to use GUI.

20 - User instance removed if program terminated without logout - Success: Removed Instance when user program is terminated - Correction procedure: open currentusers.txt and wipe the file and save this removes the instance.

21 - Download File - Success: File downloaded - Correction procedure: review file path in the application.

6.2 Test Report

1 - 23/04 - Pass
2 - 23/04 - Pass
3 - 23/04 - Pass
4 - 23/04 - Pass
5 - 23/04 - Pass
6 - 23/04 - Pass
7 - 23/04 - Pass
8 - 23/04 - Pass
9 - 23/04 - Pass
10 - 23/04 - Pass
11 - 23/04 - Pass
12 - 23/04 - Pass
13 - 23/04 - Pass
14 - 23/04 - Pass
15 - 23/04 - Pass
16 - 23/04 - Pass
17 - 23/04 - Pass
18 - 23/04 - Pass
19 - 23/04 - Pass
20 - 23/04 - Fail - Does not remove instance when program terminated without logout. This means a user will be logged out of their account if they don't log out.
21 - 23/04 - Pass

6.3 Result analysis

Only one test failed in my test plan. Each test case is related to one requirement. We know these tests passed by a success message appearing on the GUI and the user's command being executed. It is important to test the application to remove unwanted bugs in the program. This design reflects the implementation well.

7 Conclusion

In conclusion, I believe the project was a success. Our system was able to create a user, managed files, emulate a terminal and save files to a separate container. This was achieved by using JavaFX scene builder, Java, Docker and SQLite. We tested this thoroughly and all our tests passed except one. This means that the user will be able to use the majority of the features created. In future, I would like to create a better instance checker using sessions or tokens instead of a text file. Furthermore, I would like to implement more containers to make the program more scalable. However, in the time allocated this project was a success.