# CS 401 Group Project

Design Requirements Specification  
for the Distributed File System (DFS)

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Revision** | **Description** | **Author** |
| 03/15/2022 | 1.1 | Initial Version | Miles Vizinau |
| 04/04/2022 | 1.2 | Updated the Introduction Section | Miles Vizinau |
| 04/05/2022 | 1.3 | Filled out all sections of design document in preparation for phase 2 presentations | All |
| 04/16/2022 | 1.4 | Added “Log” class and updated the class diagram to match the implementation | Miles Vizinau |
| 4/26/2022 | 1.5 | Revised client section of document to match implementation | Issiah Deleon |
| 05/02/2022 | 1.6 | Updated Class Diagram, added more Architecture Diagrams | Miles Vizinau |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1. Introduction 3](#_Toc1027883213)

[1.1. Goals and Objectives 4](#_Toc2050656821)

[1.2. Scope 4](#_Toc1349025176)

[1.3. Software Context 4](#_Toc1779594142)

[1.4. Major Constraints 4](#_Toc279568261)

[1.5. References 4](#_Toc1516591387)

[2. Designs 4](#_Toc1637085722)

[2.1. Data Design 5](#_Toc558614048)

[2.2. Client Side 5](#_Toc2036991766)

[2.3. Server Side 5](#_Toc1884723813)

[2.4. Architectural and Component-Level Design 5](#_Toc672322)

[2.5. Program Structure 5](#_Toc1794641441)

[2. Classes 5](#_Toc707566126)

[2.1. Client 6](#_Toc1821813578)

[2.1.1￼Description 6](#_Toc1580745920)

[2.1.2￼Processing Narrative 6](#_Toc60214802)

[2.1.3￼Interface Description 6](#_Toc752335741)

[2.1.4￼Processing Details 6](#_Toc899131915)

[2.2. Server 6](#_Toc1371042438)

[2.2.1￼Description 6](#_Toc811656854)

[2.2.2￼Processing Narrative 6](#_Toc1878398706)

[2.2.3￼Interface Description 6](#_Toc2086378912)

[2.2.4￼Processing Details 6](#_Toc207131546)

[2.3. User 6](#_Toc1691280890)

[2.3.1￼Description 7](#_Toc822160408)

[2.3.2￼Processing Narrative 7](#_Toc2116516157)

[2.3.3￼Interface Description 7](#_Toc1541360381)

[2.3.4￼Processing Details 7](#_Toc2053007104)

[2.4. File 7](#_Toc1169003807)

[2.4.1￼Description 7](#_Toc2071310191)

[2.4.2￼Processing Narrative 7](#_Toc575312386)

[2.4.3￼Interface Description 7](#_Toc314006675)

[2.4.4￼Processing Details 7](#_Toc342220242)

[2.5. Node 7](#_Toc2070694656)

[2.5.1￼Description 7](#_Toc436472655)

[2.5.2￼Processing Narrative 7](#_Toc31994959)

[2.5.3￼Interface Description 8](#_Toc1143572833)

[2.5.4￼Processing Details 8](#_Toc1515730853)

[2.6. Log 8](#_Toc1112581985)

[2.6.1￼Description 8](#_Toc1099438987)

[2.6.2￼Processing Narrative 8](#_Toc1403894821)

[2.6.3￼Interface Description 8](#_Toc1711357811)

[2.6.4￼Processing Details 8](#_Toc332084513)

[3. Interfaces 8](#_Toc609943602)

[3.1￼Software Interface Description 9](#_Toc279642179)

[3.2￼External Interfaces 9](#_Toc1258441010)

[3.3￼Internal Interfaces 9](#_Toc679168978)

[3.4￼Human Interfaces 9](#_Toc1250043380)

[3.5￼User Interface Design 9](#_Toc1562681403)

# Introduction

This document outlines the design requirements for the Distributed File System (DFS)

## Goals and Objectives

This document describes important aspects of the implementation of the Server, Client, User, Node, File, Hidden and Unhidden storage modules.

## Scope

This document emphasizes the “*how”* of the distributed file system and the interaction between the various classes. It will not go into detail on the code implementation of the system. Priorities include the usability and simplicity of the distributed file system.

## Software Context

Users will log in to the system to utilize the DFS. After a user has logged in, they can then submit files to the DFS, or request a copy of the file by name. The client module contains the methods for LogIn() and LogOut(), which will manage the log in process. There will be two storage modules, a hidden storage and an unhidden storage. The hidden storage module will contain files that are only available to a supervisor (with a valid supervisor ID), and the unhidden storage module will contain files that are available to all users, supervisor or not. In addition to the creation of new user accounts, the supervisors will hold privileges to delete files from the entire system, remove users, and view an event history log.

## Major Constraints

1. Issue: How should files be stored in regards to there being hidden and unhidden files
   1. Each file has a flag signifying whether it’s hidden or not
   2. Have 2 different storages, 1 for hidden & 1 for unhidden
2. Decision: In order to avoid the issue of a user changing the flag on a file, which would mess with the file’s access level, we will use choose option “b”
3. Issue: We will not be storing passwords and usernames in the system for the sake of simplicity. We assume that if a user is using the system, that they have the right to use it. The management of security and passwords will be taken care of by the company. Any user can create a new account (with a password and ID).
4. Issue: How will we handle version control?
   1. If user retrieves file A from system and proceeds to edit it, and upload a new version, then a new file B is uploaded.
   2. If a user retrieves file A from system and proceeds to edit and upload, then file A is overwritten
5. Decision: For simplicity’s sake, the system will create a new file when an old version has been edited and reuploaded.

## References

# Designs

## Data Design

The DFS will manage files of any type. It will not manipulate the content of the files, but rather serve as an organizational system for all of the company’s files. We will use a simple list to keep track of all files in both the hidden and unhidden file systems, organized by chronological order .

## Client Side

Client has methods such as taskManager() and getRequestType() which manage the input/output of gathering the request type. Client prompts user for system name, username and password, and determines whether the user already exists in the system. The client sets the request object attributes accordingly and then sends these objects to the server, after which it waits for a request object sent back from the server as a response.

## Server Side

The server has methods for handling the different requests from the request object such as logging the user out, adding a file to an existing file list, removing a file, viewing the event history, clearing the event history, and shutting down the server. These requests are handled appropriately according to the user’s credentials. After every request is handled, the event appended to an existing list of events.

## Architectural and Component-Level Design

See class diagrams for a better understanding and visualization of the system

architecture.

## Program Structure

The DFS runs as a client server application, where the user will log in to the client (the instance of the software on the computer), which will then communicate with the server when files are uploaded or retrieved. There will be multiple instances of a client, but only one instance of the server.

# Classes

## Client

## 2.1.1 Description

* The client class will be responsible for acting as the mediator between the user and the server
* Will contain all the methods to communicate with the server, as well as gathering input from the user (such as system name, username/password, request type)

## 2.1.2 Processing Narrative

* Handles user logging in and out
* Handles closing the server
* Handles requesting file operations from the server

## 2.1.3 Interface Description

* Keyboard and mouse

## 2.1.4 Processing Details

* As soon as the client is opened, a log in prompt is displayed
  + The user can’t proceed further with client operations until they log in
* The client displays a GUI for the user to interact with
  + Contains buttons for the different methods contained in the client class

## Server

## 2.2.1 Description

* The server class will be responsible for having a client handler subclass which handles requests

## 2.2.2 Processing Narrative

* Handles listening for incoming client requests and creating new threads
* Handles shutting itself down

## 2.2.3 Interface Description

* Keyboard and mouse

## 2.2.4 Processing Details

* Server listens to connections from incoming clients
* As soon as it makes a connection it creates a new thread so the server can handle multiple clients (the client handler)

**2.2.b** **Client Handler**

## 2.2.1 Description

* The client handler class will be responsible for keeping a list of existing files as well as a list of events.
* Will contain methods for modifying file list
* Will contain all the methods for modifying the log list

## 2.2.2 Processing Narrative

* Handles logging client out
* Handles keeping a list of files in the system
* Handles carrying out file requests from the client
* Handles adding to the log list
* Handles carrying out log requests from the client

## 2.2.3 Interface Description

* Keyboard and mouse

## 2.2.4 Processing Details

* Performs or denies client requests
  + Actions performed are added to the event history (log list)

## User

## 2.3.1 Description

* The user class will be responsible for starting the client software and representing whether the user is a supervisor or not
* Will contain methods for opening and closing the client

## 2.3.2 Processing Narrative

* Handles opening and closing the client
* Handles retrieving information about whether the user is a supervisor or not

## 2.3.3 Interface Description

* Keyboard and mouse

## 2.3.4 Processing Details

* The user displays an interactable GUI
  + Contains buttons for the opening and closing of the client

## File

## 2.4.1 Description

* The file class will be responsible for acting as a container for any information employees which to store for the company
* Will contain methods for changing important info about said file

## 2.4.2 Processing Narrative

* Handles naming of the file
* Handles containing the information of where the file is to be stored physically
* Handles containing the size information of the file
* Handles containing the type of file that it is

## 2.4.3 Interface Description

* Keyboard and mouse

## 2.4.4 Processing Details

* Each method provides a GUI for entering the information related to the file

## Node

## 2.5.1 Description

* The node class will be responsible for acting as the physical container for the User and storage
* Storage is divided into 2 different types, each represented by a different class
  + Hidden
  + Unhidden
* Will contain a method for choosing a user to continue as

## 2.5.2 Processing Narrative

* Handles setting up basic info about the computer
  + Name, storage size, current user

## 2.5.3 Interface Description

* Keyboard and mouse

## 2.5.4 Processing Details

* As soon as the node (computer) is turned on, an interactable GUI will be displayed for setting the name and storage size of the computer
* Once the “SetCurrentUser()” method is called, an interactable GUI is displayed to pick a user to proceed as

## Log

## 2.6.1 Description

* The log class will be responsible for keeping information about everything that happens with the server
* Information is kept in 2 parts
  + The actual action that happened, such a file being deleted
  + The time associated with it, when the action actually happened
* Only supervisors can access the log

## 2.6.2 Processing Narrative

* Handles information methods
  + Adding to log
  + Clearing information form log
  + Returning log string for client to view

## 2.6.3 Interface Description

* Keyboard and mouse

## 2.6.4 Processing Details

* As soon as an action involving the server takes place, the log is automatically updated with said information
  + These “actions” are defined as the methods within the “server” class

**2.7.** **Request**

**2.7.1** **Description**

* + - The request class will be responsible for holding all of the information that is sent to and from the server.
    - Includes:
      * Node
      * Log
      * File
      * Filename
      * Filetype
      * Request type
      * Boolean indicating if it’s a hidden file
      * Boolean indicating if the client is logged in
      * Boolean indicating whether or not to print an error message
      * Boolean indicating whether or not the request was successfully processed

**2.7.2 Processing Narrative**

* + - Handles holding and extracting information:
      * Get/Set Node
      * Get/Set Log
      * Get/Set Filename
      * Get/Set Filetype
      * Get/Set Request type
      * Get/Set Hidden
      * Get/Set Logged in
      * Get/Set Error Message Boolean
      * Get/Set Request status (successful/ non-successful)

**2.7.3 Interface Description**

* Keyboard and mouse

**2.7.4 Processing Details**

* + - The client and server access the getter/setter methods within the request class. This information is what is sent back and forth between the client and server.

# Interfaces

## 3.1 Software Interface Description

- The company computers will have Windows OS as well as the client software preinstalled.

## 3.2 External Interfaces

- The hard drives stored inside the nodes will hold files

## 3.3 Internal Interfaces

- The client software will provide the user with a GUI. Upon starting the client software, the user is prompted for a user id and password in order to sign up. Once signed up, the user is assumed to be logged in and can access the server. It is at this point that the client will prompt the user for a request type depending on the user’s credentials. Every user will have access to features such as adding files, receiving files, and logging out. However, only supervisors will have access to features such as viewing the event history, clearing the event history, removing files, and shutting down the server.

## 3.4 Human Interfaces

- The user will interact with the system using company computers that have the client software preinstalled on them.

## 3.5 User Interface Design

- The client software will provide the user with a GUI that provides the user with a series of buttons and text boxes that take user input.

**Architecture Diagram(s)**





