**FILES STORAGE MANAGEMENT SYSTEM**

High Level Design & Low Level Design

The purpose of this project is to create a File Storage and Transfer System and also to layout the important High level and Low level designs.

**Document Control :**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Project Revision History** | | | | | | | | |
|  |  |  | |  |  |  |  |  |
| **Date** | **Version** | **Author** | **Brief Description of Changes** | | | | **Approver Signature** | |
| 16/11/2022 | Version 1 | Group 8 |  | | | | Trainer | |
| 24/11/2022 | Version 2 | Group 8 |  | | | | Trainer | |
|  |  |  |  | | | |  | |

[**1. Introduction**](#_heading=h.3znysh7) **5**

[1.1. Intended Audience](#_heading=h.2et92p0) 5

[1.2. Acronyms/Abbreviations](#_heading=h.tyjcwt) 5

[1.3. Project Purpose](#_heading=h.3dy6vkm) 5

[1.4. Key Project Objectives](#_heading=h.1t3h5sf) 5

[1.5. Project Scope and Limitation](#_heading=h.4d34og8) 5

[*1.5.1. In Scope*](#_heading=h.2s8eyo1) 6

[*1.5.2. Out of scope*](#_heading=h.17dp8vu) 6

[1.6. Functional Overview](#_heading=h.3rdcrjn) 6

[1.7. Assumptions, Dependencies & Constraints](#_heading=h.26in1rg) 6

[1.8. Risks](#_heading=h.lnxbz9) 6

[**2. Design Overview**](#_heading=h.35nkun2) **7**

[2.1. Design Objectives](#_heading=h.1ksv4uv) 8

[*2.1.1. Recommended Architecture*](#_heading=h.44sinio) 8

[2.2. Architectural Strategies](#_heading=h.2jxsxqh) 8

[*2.2.1. Design Alternative*](#_heading=h.z337ya) 8

[*2.2.2. Reuse of Existing Common Services/Utilities*](#_heading=h.3j2qqm3) 8

[*2.2.3. Creation of New Common Services/Utilities*](#_heading=h.1y810tw) 9

[*2.2.4. User Interface Paradigms*](#_heading=h.4i7ojhp) 9

[*2.2.5. System Interface Paradigms*](#_heading=h.2xcytpi) 9

[*2.2.6. Error Detection / Exceptional Handling*](#_heading=h.1ci93xb) 9

[*2.2.7. Memory Management*](#_heading=h.3whwml4) 9

[*2.2.8. Performance*](#_heading=h.2bn6wsx) 9

[*2.2.9. Security*](#_heading=h.qsh70q) 9

[*2.2.10. Concurrency and Synchronization*](#_heading=h.3as4poj) 9

[*2.2.11. Housekeeping and Maintenance*](#_heading=h.1pxezwc) 9

[**3. System Architecture**](#_heading=h.49x2ik5) **10**

[3.1. System Architecture Diagram. (Not Necessary)](#_heading=h.2p2csry) 11

[3.2. System Use-Cases](#_heading=h.147n2zr) 12

[3.3. Subsystem Architecture](#_heading=h.3o7alnk) 13

[3.4. System Interfaces](#_heading=h.23ckvvd) 14

[*3.4.1. Internal Interfaces*](#_heading=h.ihv636) 14

[*3.4.2. External Interfaces*](#_heading=h.32hioqz) 14

[**4. Detailed System Design**](#_heading=h.1hmsyys) **14**

[4.1. Key Entities](#_heading=h.41mghml) 14

[4.2. Detailed-Level Database Design](#_heading=h.2grqrue) 14

[*4.2.1. Data Mapping Information 1*](#_heading=h.vx1227)4

[*4.2.2. Data Conversion 1*](#_heading=h.3fwokq0)4

[4.3. Archival and retention requirements](#_heading=h.1v1yuxt) 14

[4.4. Disaster and Failure Recovery](#_heading=h.4f1mdlm) 15

[4.5. Business Process workflow](#_heading=h.2u6wntf) 15

[4.6. Business Process Modeling and Management (as applicable)](#_heading=h.19c6y18) 15

[4.7. Business Logic](#_heading=h.3tbugp1) 15

[4.8. Variables](#_heading=h.28h4qwu) 15

[4.9. Activity / Class Diagrams (as applicable)](#_heading=h.nmf14n) 15

[4.10. Data Migration](#_heading=h.37m2jsg) 15

[*4.10.1. Architectural Representation*](#_heading=h.1mrcu09) 15

[*4.10.2. Architectural Goals and Constraints*](#_heading=h.46r0co2) 15

[*4.10.3. Logical View*](#_heading=h.2lwamvv) 15

[*4.10.4. Architecturally Significant Design Packages*](#_heading=h.111kx3o) 15

[*4.10.5. Data model*](#_heading=h.3l18frh) 15

[*4.10.6. Deployment View*](#_heading=h.1egqt2p) 16

[**5. Environment Description**](#_heading=h.3ygebqi) **16**

[5.1. Time Zone Support](#_heading=h.2dlolyb) 16

[5.2. Language Support](#_heading=h.sqyw64) 16

[5.3. User Desktop Requirements](#_heading=h.3cqmetx) 16

[5.4. Server-Side Requirements](#_heading=h.1rvwp1q) 16

[*5.4.1. Deployment Considerations*](#_heading=h.4bvk7pj) 16

[*5.4.2. Application Server Disk Space*](#_heading=h.2r0uhxc) 16

[*5.4.3. Database Server Disk Space*](#_heading=h.1664s55) 17

[*5.4.4. Integration Requirements*](#_heading=h.3q5sasy) 17

[*5.4.5. Jobs*](#_heading=h.25b2l0r) 17

[*5.4.6. Network*](#_heading=h.kgcv8k) 17

[*5.4.7. Others*](#_heading=h.34g0dwd) 17

[5.5. Configuration](#_heading=h.1jlao46) 17

[*5.5.1. Operating System*](#_heading=h.43ky6rz) 17

[*5.5.2. Database*](#_heading=h.2iq8gzs) 17

[*5.5.3. Network*](#_heading=h.xvir7l) 17

[*5.5.4. Desktop*](#_heading=h.3hv69ve) 17

[**6. References**](#_heading=h.1x0gk37) **17**

[**7. Appendix**](#_heading=h.4h042r0) **18**

# 

# Introduction

## File Storage and Management System is one of the most crucial elements in a data-centric environment. Since, data has become integral in the day to day operations of organizations. It is imperative to create a robust data and file management system.

## File transfer is the process of copying or moving a file from one computer to another over a network or Internet connection. The basic idea is to create a server that listens on a particular port, this server will be responsible for receiving files (you can make the server send files as well). On the other hand, the client will try to connect to the server and send a file of any type.

## We intend to develop a File Storage system which is robust and secure, we utilize signals as well in this project to utilize the project functionality.

## 1.1 Intended Audience

|  |  |
| --- | --- |
| BU Authority |  |
|  |  |

## 1.2 Acronyms/Abbreviations

|  |  |
| --- | --- |
| DBMS | Database Management System |
| IP | Internet Protocol |
| IPv4 | Internet Protocol version 4 |
| IPv6 | Internet Protocol version 6 |
| TCP | Transfer Control Protocol |

## 1.3 Project Purpose

The purpose of this project is to get a solid grasp on the fundamentals of the Socket API, File Transfer using Protocols. Writing such an application in C++ gives a basic understanding on how the client - server architecture works and overall on how to use the Socket API to establish communication between client and server applications. We also intend to understand the transfer and storing of files through this project.

## 1.4 Key Project Objectives

The objective is to create a File Transfer feature, which allows the clients to store data and also access it at their own accord. Data integrity and security is preserved for the client and allows them to provide access to other clients if needed.

## 1.5 Project Scope and Limitation

SCOPE:

The scope of the project includes that a communication must be formed between the client and the server through which the client can transmit their data. This data sent to a server will be stored and processed. Data retrieval can also be performed in this project.

LIMITATION:

* Multiple clients can connect to only one server.

### 1.5.1 In Scope

* Clients provide data to the server.
* Servers and clients work in tandem and store information.
* File retrieval and File transfer features can be executed

### 1.5.2 Out of scope

* Multiple Clients can give multiple data and files to the server for storage.

## 1.6 Functional Overview

**Client:**

In the client terminal the client will either do registration or login. After successful login, the client will send their data to the server, the client also has the functionality to share the access of the data if needed. The client is also given a catalogue of options which allows the client

**Server:**

In the server terminal the server first login with its credentials after that it will receive the request from the client. After that request, the server communicates and stores the data of the client.

## 1.7 Assumptions, Dependencies & Constraints

OPERATING SYSTEMS:

Operating environment for implementing File Storage and Transfer Management are:

* Client/server system
* Operating system: Linux
* Platform: Ubuntu/C++

## 1.8 Risks

No Risk(As it is for educational purpose)

# Design Overview

1. Start:

This is the start block which indicates the start of the program.

1. Client Authentication:

This is the module used for the client login where the client, if new, has to register with an username and password and then login by entering the same credentials (username and password). If already registered the client can login with the credentials. If the client used invalid credentials for login then it will show some error message like “Invalid Credentials”.`

1. Upload Files:

After successful validation, client will be connected to the server’s port number. Once connected to the server’s socket this module enables client to upload new files into their respective directories.

1. Download Files:

This module enables client to download files from server after successful validation of their credentials.

1. Display Respective User’s Data:

In this module client can able to the see list of files in their respective directories.

1. Delete Files:

In this module client can be able to delete the files in their directories . The deleted file details updated in their directories and shows the reformed list.

1. Server:

This module creates socket and binds with a port number. After binding Server for connecting with multiple clients (having same port number as server). After connection get formed with client, Server process the requests of clients and provide the response according to their request(upload/download/delete).

1. Exit:

This ensures that the program has terminated.

## 2.1 Design Objectives

* Create login credential page for both server and client
* After successful login, the client is given a menu which contains plethora of options
* Options include upload, download, display and delete
* Once the required option is selected, the data is relayed to the server
* Server parses the data and sends it back to the client for display.

### 2.1.1 Recommended Architecture

Generic

## 2.2 Architectural Strategies

* Header files
* Structures
* Macros

### 2.2.1 Design Alternative

NA

### 2.2.2 Reuse of Existing Common Services/Utilities

### 2.2.3 Creation of New Common Services/Utilities

NA

### 2.2.4 User Interface Paradigms

Command Line Interface: Terminal

### 2.2.5 System Interface Paradigms

Command Line Interface: Terminal

### 2.2.6 Error Detection / Exceptional Handling

Error Detection:

1. Invalid user login
2. Invalid File being sent
3. File not found asked in retrieval
4. Error in establishing connection between server and client

### 2.2.7 Memory Management

NA

### 2.2.8 Performance

NA

### 2.2.9 Security

For security purposes the system asks for login credentials from server and client.

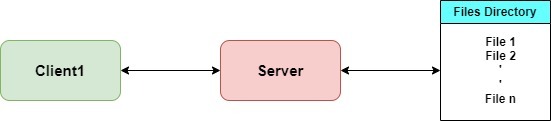
### 2.2.10 Concurrency and Synchronization

NA

### 2.2.11 Housekeeping and Maintenance

NA

# 3. System Architecture

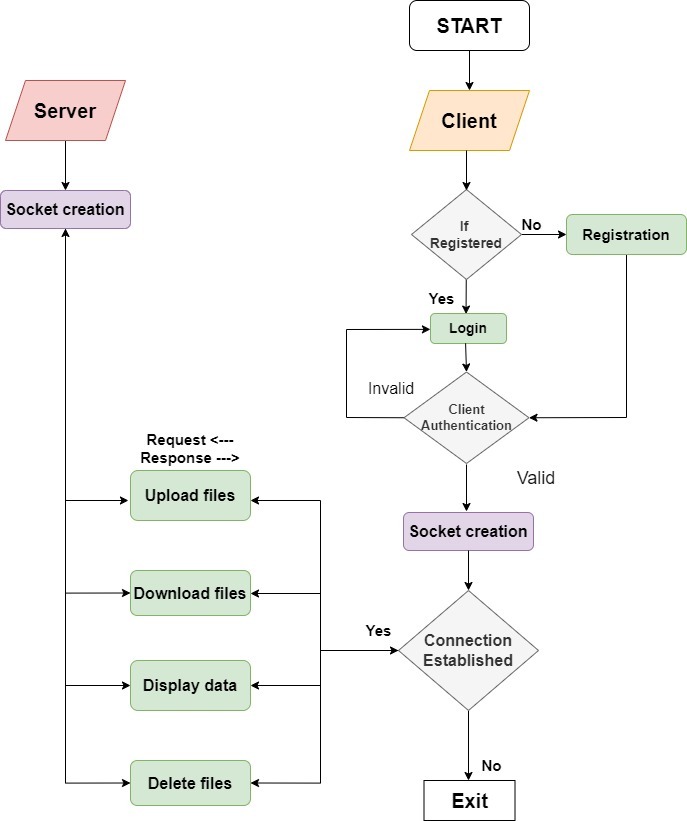
****



**LEVEL 0 DFD:**

**LEVEL 1 DFD:**

## 3.1 System Architecture Diagram. (Not Necessary)



## 3.2 System Use-Cases

**

*Source: geeksforgeeks*

## 3.3 Subsystem Architecture

NA

## 3.4 System Interfaces

NA

### 3.4.1 Internal Interfaces

NA

### 3.4.2 External Interfaces

NA

# Detailed System Design

The code starts by declaring the login credentials for the client. Once the client is logged into the system. The communication between server and client is established, which facilitates file transfer.

Using Socket (TCP) the client is able to share data to the server which the server. With this communication between the server-client architecture, the client is able to send their data to the necessary server for storage and retrieval.

## 4.1 Key Entities

* Valid login details for client
* IP Address
* Valid Files to be sent and received

## 4.2 Detailed-Level Database Design

NA

### 4.2.1 Data Mapping Information

Mapping the client side data in the directories, for easy access and retrieval

## 4.3 Archival and retention requirements

NA

## 4.4 Disaster and Failure Recovery

* We don’t have any control over the system. In case of failure, source code is safe.
* Use of Git.

## 4.5 Business Process workflow

NA

## 4.6 Business Process Modeling and Management (as applicable)

NA

## 4.7 Business Logic

NA

## 4.8 Variables

NA

## 4.9 Activity / Class Diagrams (as applicable)

**CLIENT;**

/\*Start: Show the Menu for login, register and exit

option->takes user choice\*/

STEP 1:

start

{ print(select an option:

1. register

2. login

3. exit)

}

STEP 2:

switch(option){

case 1: Login : /\*It takes Existing valid username and password as inputs

otherwise pop ups error\*/

Login{

print("welcome to login")

print("enter user name : ")

scan(user\_name)

Read\_File(Users.txt)

if( user\_name != exists)

{

print("Opps! user not exist. Please enter a valid username")

Repeat STEP 1

}

else{

print("enter password : ")

scan(pass\_word)

Read\_File(Users.txt)

If(user\_name & pass\_word != exists)

{

print("Sorry, Enter a valid Password")

}

else{

print("login in Successfully")

continue;

}

}

case 2: Register: /\*It takes username and password for new users\*/

Register{

print("enter user name : ")

scan(user\_name)

if( user\_name == exists)

{

print("user is already registered

Try again with different user name")

Repeat STEP 1

}

else{

print("enter password to register : ")

scan(pass\_word)

Write\_to\_file(user\_name, pass\_word)

}

case 3: Exit:/\*Terminate Program......\*/

Exit{

print("Thank you for choosing us!

Have a nice day........!!")

}

}

----------------------------------------------------------------------------------

----------------------------------------------------------------------------------

void send\_file(int sockfd,FILE \*fp,char filename)

{

send(filename)

while(true){ /\*An infinite loop will run until entire file received\*/

read(data,fp) /\*read each line from file\*/

if(data != NULL)

{

send\_status=send(data)

if(send\_status<0)

print("Error in sending the file")

return

}

else

print("File sended successfully");

return

}

}

void write\_file(int sockfd)

{

recv(filename)

Intialise file\_pointer fptr /\*intialise and open a file\*/

while(true)

{

read\_status=recv(data)

if(read\_status<0)

break;

write\_to\_file(data)

}

print("Download Completed")

}

----------------------------------------------------------------------------

----------------------------------------------------------------------------

connect()

/\*Connection forms with server using Server's port Address.

If connection fails,print error message "Connection failed."\*/

Connect()

{

create clientsocket;/\*using socket()\*/

if(clientsocket < 0)

{ print("creation of client socket failed") }

if(connect(client socket, server\_address, sizeof(server\_address)<0)

{

print("connection error")

}

else

{

print("connection established")

}

}

Show menu

1.Upload file/\*to upload enter signal ctrl+c\*/

2.Download file/\*to download enter signal ctrl+z\*/

3.Delete file/\*to Delete enter signal ctrl+d\*/

4.Display /\*to display enter signal ctrl+f\*/

option-> Takes user choice

switch(option)

{

case 1: upload()

{

send\_to\_server("upload")/\*send upload message to server\*/

scan(filename)

FILE \*fptr=open(filename)

If(fptr\_open())

goto send\_file(sockfd,fptr,filename)

else

print("File cannot be open.")

}

case 2: download()

{

send\_to\_server("download")/\*send download message to server\*/

goto write\_file(sockfd);

}

case 3: delete()

{

send filename to server.

If file doesn't exist, print "File not found!"

else print "File deletion Succesful".

}

case 4: display():list the file names of particular user

}

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

**Server:**

main()

{

create serverSocket;/\*using socket()\*/

if(serverSocket < 0)

{ print("creation of client socket failed")

}

bind\_status = bind(ServerSocket,port)/\*binds the server socket with particular port number\*/

if(bind\_status<0)

{

print("Socket binding has failed")

}

listen\_status=listen(serverSocket,5) /\*listen for incoming connection requests from client\*/

if(listen\_status<0)

print("Listening faied")

else

print("....waiting for connectioms....")

if(connect(client socket, server\_address, sizeof(server\_address)<0)

{

print("connection error")

}

else

{

print("connection established")

}

}

read\_status=recv\_from\_client(data)

if(data == "upload")

{

goto write\_file();

}

else if(data == "download")

{

goto send\_file();

}

delete()

{

if (file\_name==exists)

{delete filename}

else {

(print "File not found!")

}

}

## 4.10 Data Migration

NA

### 4.10.1 Architectural Representation

NA

### 4.10.2 Architectural Goals and Constraints

The project is just for educational purposes.

### 4.10.3 Logical View

NA

### 4.10.4 Architecturally Significant Design Packages

NA

### 4.10.5 Data model

NA

**Legacy system data model**

**Proposed system data model**

**Interface data model**

### 4.10.6 Deployment View

NA

# Environment Description

* **G++:** GNU C++ Compiler ( g++ ) is a compiler in Linux which is used to compile C++ programs. It compiles both files with extension .c and .cpp as C++ files.
* **Socket Programming:** Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server.
* **UBUNTU:** Ubuntu is an open-source operating system (OS) based on the Debian GNU/Linux distribution. Ubuntu incorporates all the features of a Unix OS with an added customizable GUI, which makes it popular in universities and research organizations. Ubuntu is primarily designed to be used on personal computers, although a server edition does also exist.
* **GITHUB:** GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere. This tutorial teaches you GitHub essentials like repositories, branches, commits, and pull requests.

## 5.1 Time Zone Support

NA

## 5.2 Language Support

NA

## 5.3 User Desktop Requirements

Linux, Ubuntu

## 5.4 Server-Side Requirements

Linux, Ubuntu

### 5.4.1 Deployment Considerations

NA

### 5.4.2 Application Server Disk Space

NA

### 5.4.3 Database Server Disk Space

NA

### 5.4.4 Integration Requirements

NA

### 5.4.5 Jobs

NA

### 5.4.6 Network

NA

### 5.4.7 Others

NA

## 5.5 Configuration

NA

### 5.5.1 Operating System

Linux desktop editions with 8 GB RAM- A GUI-based LINUX system must be used

### 5.5.2 Database

NA

### 5.5.3 Network

### 5.5.4 Desktop

* CPU : Intel i3/i5/i7 generation 3 and later
* RAM: 4GB or greater - For optimal performance, 6GB or 8GB are recommended if you will be running multiple browser tabs and/or multiple applications at the same time
* Internal memory:476 GB SSD/HDD.

# References

<https://man7.org/linux/man-pages/index.html>

<https://www.csd.uoc.gr/~hy556/material/tutorials/cs556-3rd-tutorial.pdf>

<https://www.ibm.com/docs/en/zos/2.2.0?topic=reference-library-functions>

# Appendix

**Change Log**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **QMS Template Version Control (Maintained by QA)** | | | | | |
|  |  |  |  |  |  |
| **Date** | **Version** | **Author** | | **Description** | |
| 23/11/22 | 1.0 | QA Team | | Initial Version | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |