File Storage System

Software Requirements Specification

Megan Paffrath, Haiyang Sun, Priyanka Gopal,
Prasoon Shakya, Monicah Gikanga

Revision History

Date	Revision	Description	Author
06/14/2020	1.0	Initial Version	Megan Paffrath, Haiyang Sun, Priyanka Gopal, Prasoon Shakya, Monicah Gikanga
06/22/2020	1.1	first update	Megan Paffrath, Haiyang Sun, Priyanka Gopal, Prasoon Shakya, Monicah Gikanga
07/07/2020	1.2	Second Update	Megan Paffrath, Haiyang Sun, Priyanka Gopal, Prasoon Shakya, Monicah Gikanga
07/15/2020	1.3	Final Version	Megan Paffrath, Haiyang Sun, Priyanka Gopal, Prasoon Shakya, Monicah Gikanga

Table of Contents

1. Purpose **4**

1.1. Scope 4
1.2. Definitions, Acronyms, Abbreviations 4
1.3. References 4
1.4. Overview 4

2. Overall Description **5**

2.1. Product Perspective 5
2.2. Product Architecture 5
2.3. Product Functionality/Features 5
2.4. Constraints 5
2.5. Assumptions and Dependencies 5

3. Specific Requirements **6**

3.1. Functional Requirements 6
3.2. External Interface Requirements 6
3.3. Internal Interface Requirements 7

4. Non-Functional Requirements **8**

4.1. Security and Privacy Requirements 8
4.2. Environmental Requirements 8
4.3. Performance Requirements 8

1. Purpose

This document outlines the requirements for the File Storage System.

1.1. Scope

This document catalogs the File Storage System's requirements for the user, system as well as hardware. This document will not cover how these requirements will be implemented.

1.2. Definitions, Acronyms, Abbreviations

FSS: File Storage System

1.3. References

Use Case Specification Document
UML Use Case Diagrams Document
UML Class Diagram

1.4. Overview

The File Storage System (FSS), is designed for users to store, download as well as share files with other users.

2. Overall Description

2.1. Product Perspective

FSS is a GUI implementation system of client/server model. This program helps users store their files to prevent the loss of data.

2.2. Product Architecture

The system will be organized into two major components, the server and the client. There will be one server running that can serve multiple clients at the same time.

2.3. Product Functionality/Features

The high-level features of the system are as follows (more detailed requirements in section 3):

Server:

- Collects and sends serialized messages to and from the clients to support functionality
- User Login: collects user information and determines if user is valid
- User Logout: disconnects client from server
- New User: create a new user
- Stores users and their files
    - Files contain version history
- Maintains logs
    - Keeps track of when users log in, make changes, and log out
- Can serve multiple clients at a time
- Supports user interactions
    - user can view all of the files they have stored on the server
    - users can view all of the files that others on the server have shared with them
    - user can upload files to server
    - user can change files on the server
    - user can grant access of files to other users
    - users can download stored files to their local system

Client:

- Collects and sends serialized messages to and from the server to support functionality
- Login: sends user credentials to server for verification of valid user
- Logout: ends interaction with the server
- Sign up: create a new user on the server
- View User Files: can view all of the files that the user has on the server
- Open Folder: can view the contents of a folder
- Read content: requests server for file contents and returns the file
- Upload Content: sends files to server to upload
- Delete Content: requests server to remove content
- Download: requests server for file(s) and returns the files to the user's system
- Share: requests server to let another user access shared files

2.4. Constraints

- Users should be able to go back to one of the older copies of their file in case of unwanted changes.
- Each user will have access to one main directory which contains all of their files as well as all of the files that the user has access to

2.5. Assumptions and Dependencies

- The user does not store a corrupted file.
- It is assumed that the user only uploads folders and text files
- It is assumed that the amount of people using the program at a time does not exceed 10,000 people
- The user cannot store files over 10 MB
- The system will depend on a database to store files

3. Specific Requirements

3.1. Functional Requirements

3.1.1. Common Requirements:

3.1.1.1. Server and clients send serialized package data back and forth for login, download, and other functionality
3.1.1.2. Files are private to the user who uploads them unless user specifies to share the files with other users

3.1.1.4. Server Requirements:

3.1.1.3. Only one server should be running
3.1.1.4. Server can support multiple clients at a time
3.1.1.5. Takes requests from client program
3.1.1.6. verifies user login with stored usernames and passwords
3.1.1.7. allows clients to make new users
3.1.1.8. allows users to log out
3.1.1.9. supports user upload, sharing, deleting and downloading of files
3.1.1.10. Logs user events/interactions
3.1.1.11. Store users name & password data for verification later when login
3.1.1.12. react to the command correctly and specify which function it needs to perform to respond to the user's request.
3.1.1.13. automatically log out if user has been timed out or no response for long period of time

3.1.2. Client Requirements:

3.1.2.1. Multiple clients can be running and accessing the server at a time
3.1.2.2. Users can log in using a username and password (both of which are alphanumeric strings between 6 and 20 characters)
3.1.2.3. Users can add, edit, delete, download, read, and share files
3.1.2.4. Users can log out
3.1.2.5. Users can create new accounts with username and password (both of which are alphanumeric strings between 6 and 20 characters)
3.1.2.6. Users can change their passwords and emails

3.2. External Interface Requirements

3.2.1. Server and client programs will never be ran on the same IP address at the time
3.2.2. No two clients can run on the same IP address

3.3. Internal Interface Requirements

3.3.1. System will need to process username and passwords and verify if user logins are valid.
3.3.2. System must keep a log of all the times that the client/user logs-in and logs-out.
3.3.3. System must keep a log for everytime that the user uploads the file.
3.3.4. System must store file data on server as well as transfer file data between server and client programs

5. Non-Functional Requirements

5.1. Security and Privacy Requirements

5.1.1. Use file versioning to prevent file corruption.
5.1.2. Users can't view another user's files unless they're shared with them.
5.1.3. Users will be timed out if they has no respond or action for a long period of time
5.1.4. If the same user is trying to login in a different location, the previous location will be automatically logged out.

5.2. Environmental Requirements

5.2.1. Users will have the client software installed on their local computers
5.2.2. Program cannot require user to install additional supporting software
5.2.3. Server and client software can be ran in different environments

5.3. Performance Requirements

5.3.1. Clients must be able to upload, access, edit, as well as download files in reasonable time. If not complete within reasonable time, send an error message.

UML Case Specification:

Use Case ID: 1
Use Case Name: LogIn to a user account
Relevant Requirements:

- 3.1.2.2

Primary Actor: User
Pre-conditions: 1) Internet Connection
2) Opened client app
Post-conditions: User is displayed with list of features that they could access
Basic Flow or Main Scenario: 1) Users initiates action by entering their username and password
2) Server confirms that the user exists and the given password matches to the username
3) Users gets access to their account
Extensions or Alternate Flows: If the username or password does not match, then display an error message and prompt the user for entering the username and password again.

Exceptions: If invalid login, show an error message displaying "Invalid login!"

Related Use Cases: Use Case ID:2, Use Case ID:3, Use Case ID:4, Use Case ID:5, Use Case ID:6

Use Case ID: 2
Use Case Name: Log Out of the user account
Relevant Requirements:

- 3.1.2.4

Primary Actor: User
Pre-conditions:1) Internet Connection
2) Opened client app
3) Logged into the user account
Post-conditions: User gets logged out of their account
Basic Flow or Main Scenario: User LogsOut by clicking on the appropriate button on the GUI
Extensions or Alternate Flows:
Exceptions:
Related Use Cases: Use Case ID:1, Use Case ID:3, Use Case ID:4, Use Case ID:5, Use Case ID:6

Use Case ID: 3
Use Case Name: Add a File
Relevant Requirements:

- 3.1.2.3

Primary Actor: User
Pre-conditions: 1) Internet Connection
2) Opened client app
3) Logged into the user account
Post-conditions: 1) Server allocates the space for the user's data and stores it into the server's storage
Basic Flow or Main Scenario: 1) User initiates the "Add" feature by clicking on the appropriate button on the GUI
2) Users can then browse and select the file they would want to store in the server's storage
3) User sends the data to the server
4) Server stores the data in appropriate location/ directory
Extensions or Alternate Flows: If User wants to terminate the process then, they could initiate it by clicking on the "cancel" button
Exceptions: 1) File size is too big
2) Corrupted files
Related Use Cases: Use Case ID:1, Use Case ID:2, Use Case ID:4, Use Case ID:5

Use Case ID: 4
Use Case Name: Delete a File
Relevant Requirements:

- 3.1.2.3

Primary Actor: User
Pre-conditions: 1) Internet Connection
2) Opened client app
3) Logged into the user account
Post-conditions: Remove the file from the server storage
Basic Flow or Main Scenario: 1) User initiates the "Delete" feature by clicking on the appropriate button on the GUI
2) Users can then browse and select the file they would want to delete from the server storage
3) User requests the selected data to be removed from the storage
4) Server removes that particular data/file from the location/ directory
Extensions or Alternate Flows: If User wants to terminate the process then, they could initiate it by clicking on the "cancel" button
Exceptions:
Related Use Cases: Use Case ID:1, Use Case ID:2, Use Case ID:3, Use Case ID:5

Use Case ID: 5
Use Case Name: Share a File
Relevant Requirements:

- 3.1.2.3

Primary Actor: User
Pre-conditions: 1) Internet Connection
2) Opened client app
3) Logged into the user account
Post-conditions: Grants access for other users to have access to their file
Basic Flow or Main Scenario: 1) User initiates the "Share" feature by clicking on the appropriate button on the GUI
2) Users can then browse and select the file they would want to share with other users
3) User sends request to the server for granting other users to access their data/ file
4) Server then implies the access grant as per the request of the user
Extensions or Alternate Flows: If User wants to terminate the process then, they could initiate it by clicking on the "cancel" button
Exceptions:
Related Use Cases: Use Case ID:1, Use Case ID:2, Use Case ID:3, Use Case ID:4

Use Case ID: 6
Use Case Name: Create new user account
Relevant Requirements:

- 3.1.2.5

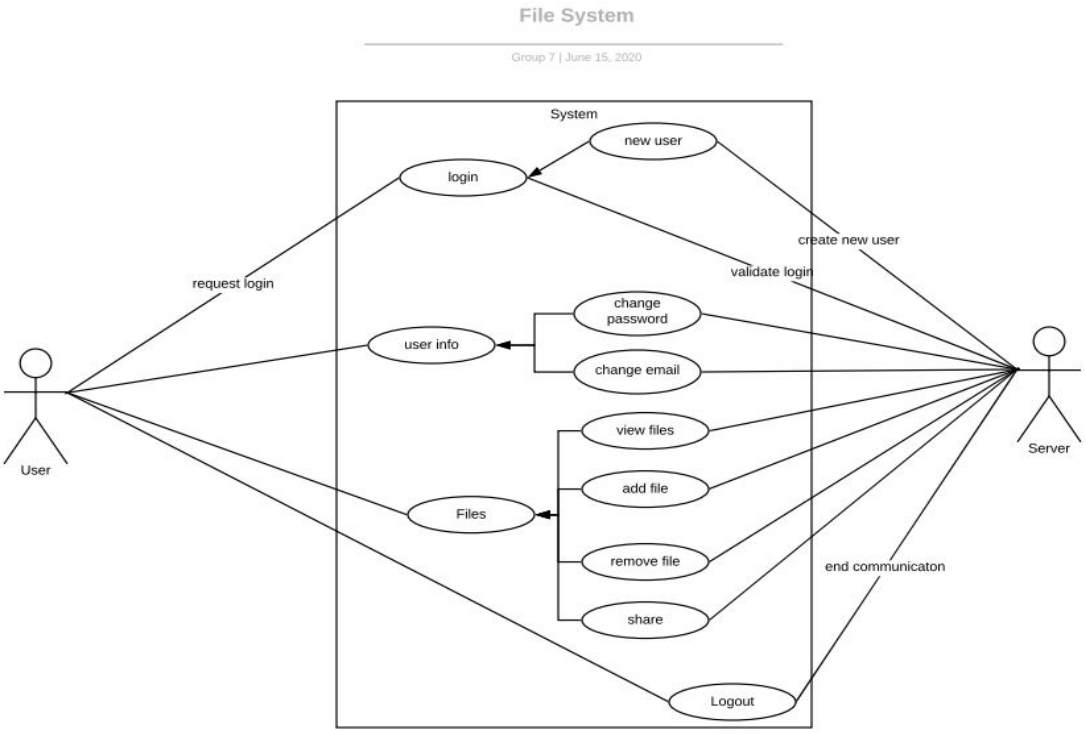
Primary Actor: User
Pre-conditions: 1) Internet Connection
2) Opened client app
Post-conditions: User is logged into the new account
Basic Flow or Main Scenario: 1) User initiates this process by clicking on Create new account button on the gui
2) Server accepts the request and asks to enter a unique username and password
3) User then has a new account
Extensions or Alternate Flows: 1) The username entered by the client has already been used, so the client will be asked to enter another unique username
2) The password does not meet the requirement, so client will be asked to enter another unique password
Exceptions:
Related Use Cases: Use Case ID:2, Use Case ID:3, Use Case ID:4, Use Case ID:5

Use Case ID: 7
Use Case Name: Change Password of the user account
Relevant Requirements:

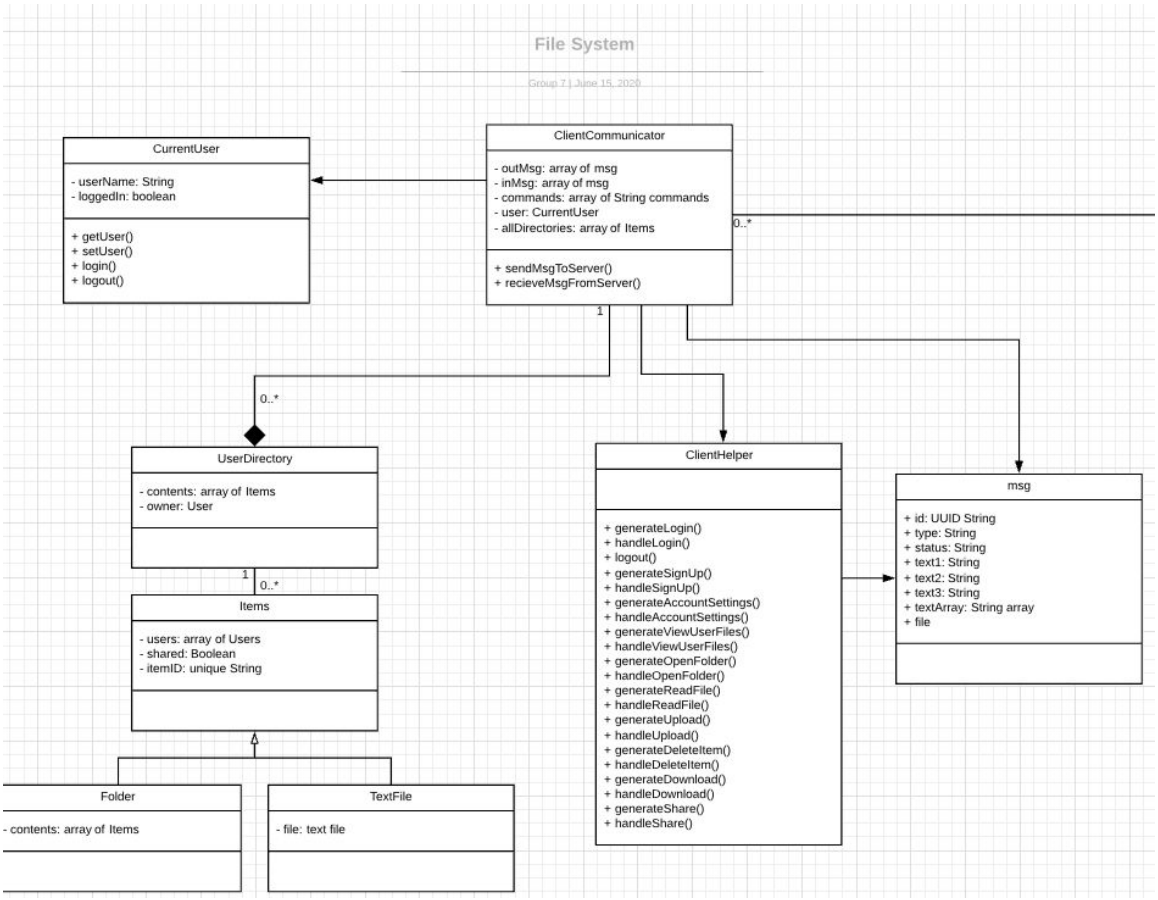
- 3.1.2.6

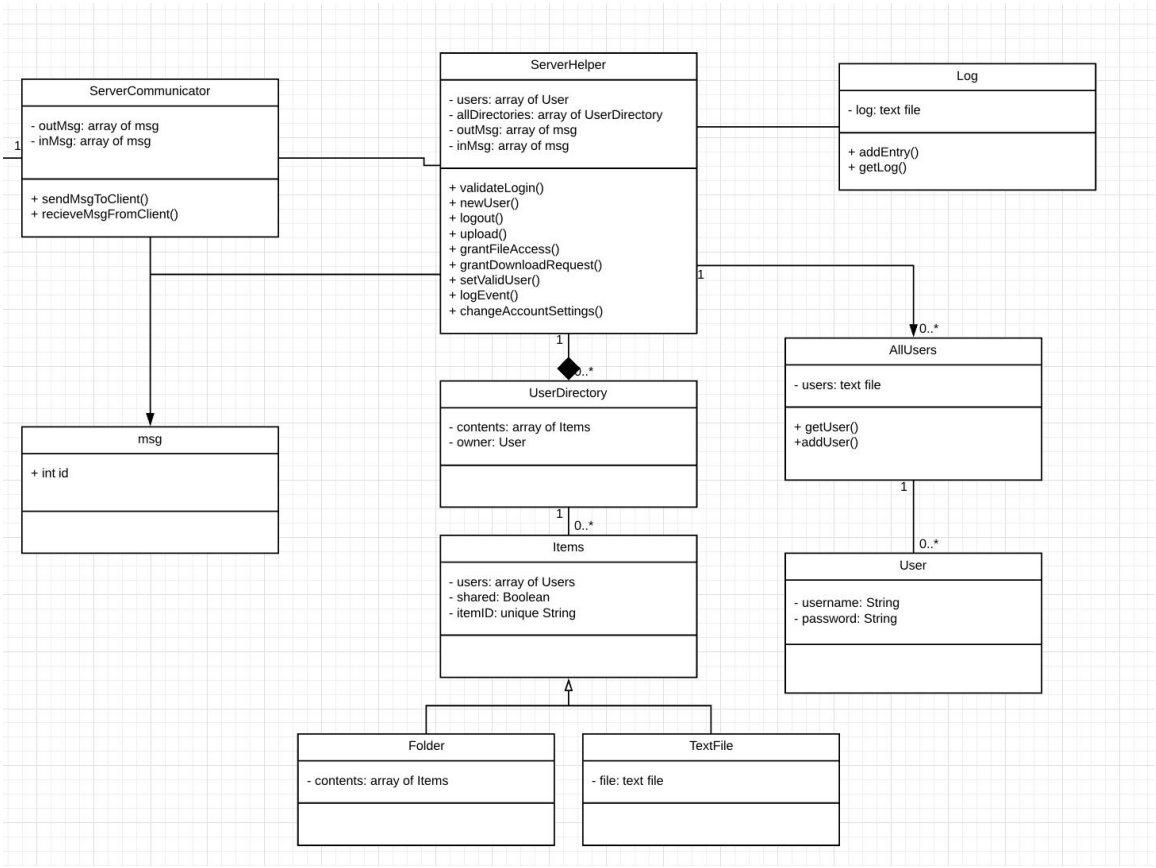
Primary Actor: User
Pre-conditions: 1) Internet Connection
2) Opened client app
3) Logged in to the user account
Post-conditions: User password has been changed
Basic Flow or Main Scenario: 1) User initiates this process by clicking on “Change password” button on the gui
2) Server accepts the request and asks to enter the current password for verification
3) Then server asks user to input new password
4) User confirms the new password by pressing “confirm” button
Extensions or Alternate Flows: 1) The current password does not match
2) terminates the process and taken to home page
Exceptions:
Related Use Cases: Use Case ID:2, Use Case ID:3, Use Case ID:4, Use Case ID:5

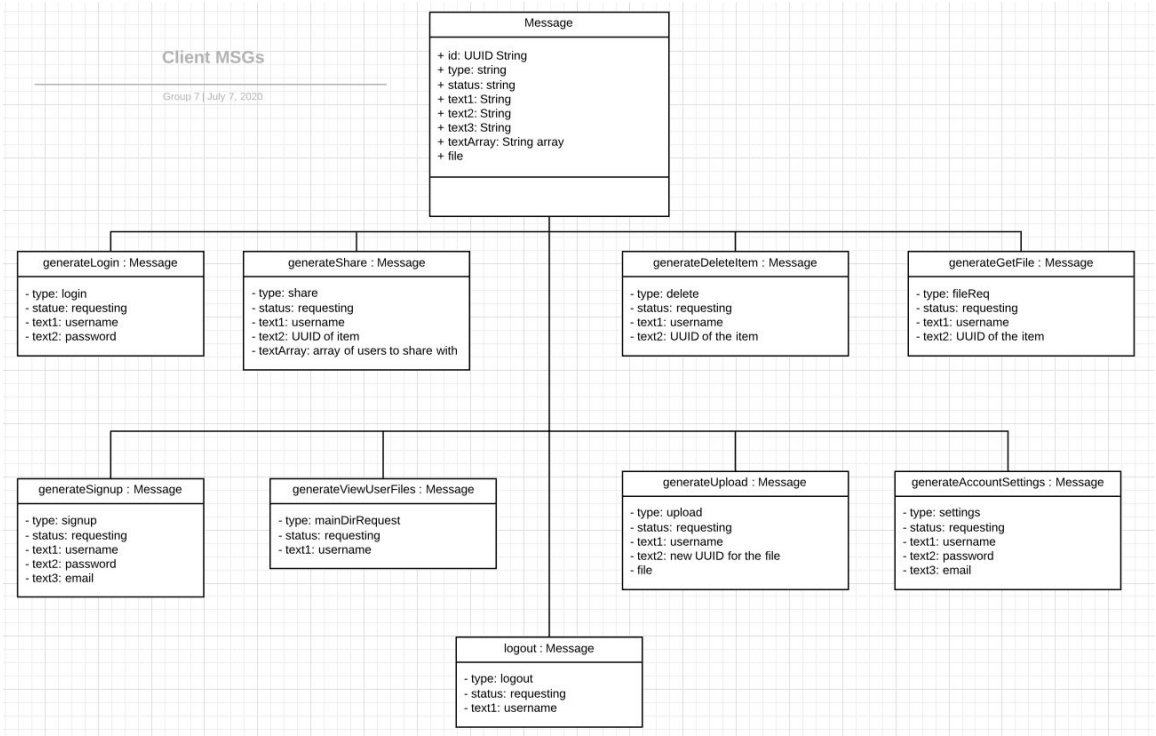
UML Case Diagram:

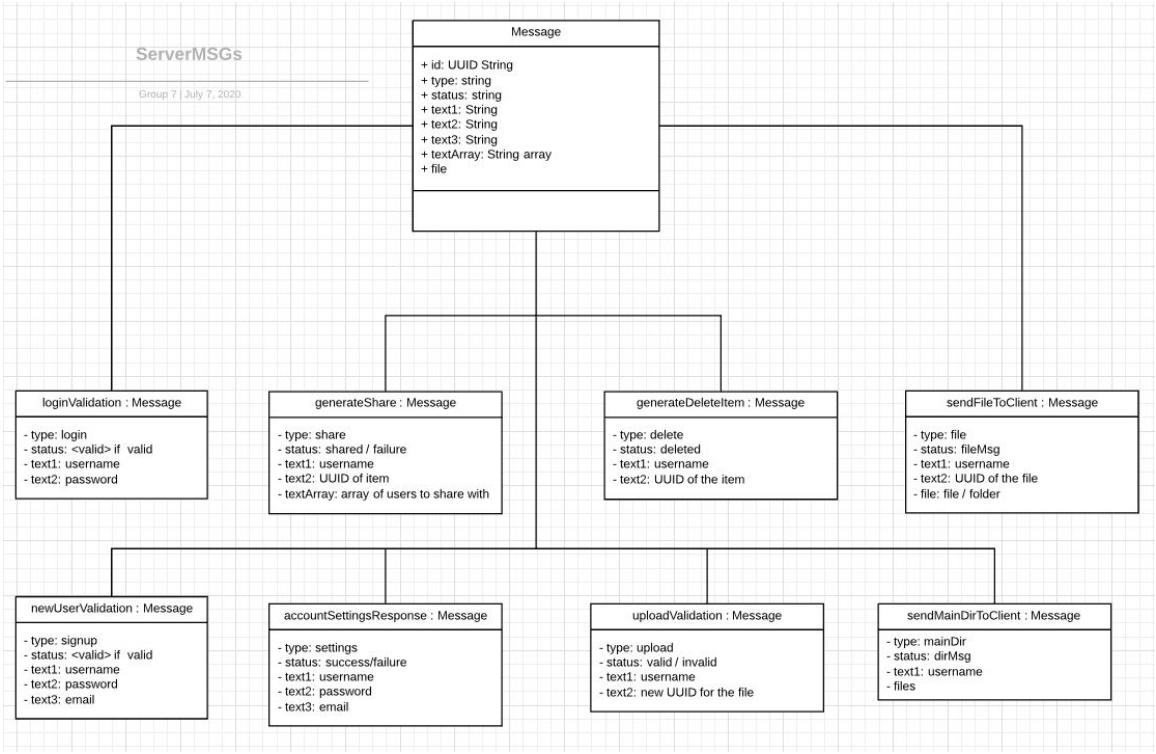


UML Class Diagram




ServerCommunicator
- outMsg: array of msg
- inMsg: array of msg
+ sendMsgToClient()
+ recieveMsgFromClient()
ServerHelper
- users: array of User
- allDirectories: array of UserDirectory
- outMsg: array of msg
- inMsg: array of msg
+ validateLogin()
+ newUser()
+ logout()
+ upload()
+ grantFileAccess()
+ grantDownloadRequest()
+ setValidUser()
+ logEvent()
+ changeAccountSettings()
Log
- log: text file
+ addEntry()
+ getLog()
msg
+ int id
UserDirectory
- contents: array of Items
- owner: User
AllUsers
- users: text file
+ getUser()
+addUser()
Items
- users: array of Users
- shared: Boolean
- itemID: unique String
User
- username: String
- password: String
Folder
- contents: array of Items
TextFile
- file: text file
1
0..*


Client MSGs
Group 7 | July 7, 2020
Message
+ id: UUID String
+ type: string
+ status: string
+ text1: String
+ text2: String
+ text3: String
+ textArray: String array
+ file
generateLogin : Message
- type: login
- statue: requesting
- text1: username
- text2: password
generateShare : Message
- type: share
- status: requesting
- text1: username
- text2: UUID of item
- textArray: array of users to share with
generateDeleteItem : Message
- type: delete
- status: requesting
- text1: username
- text2: UUID of the item
generateGetFile : Message
- type: fileReq
- status: requesting
- text1: username
- text2: UUID of the item
generateSignup : Message
- type: signup
- status: requesting
- text1: username
- text2: password
- text3: email
generateViewUserFiles : Message
- type: mainDirRequest
- status: requesting
- text1: username
generateUpload : Message
- type: upload
- status: requesting
- text1: username
- text2: new UUID for the file
- file
generateAccountSettings : Message
- type: settings
- status: requesting
- text1: username
- text2: password
- text3: email
logout : Message
- type: logout
- status: requesting
- text1: username


ServerMSGs
Group 7 | July 7, 2020
Message
+ id: UUID String
+ type: string
+ status: string
+ text1: String
+ text2: String
+ text3: String
+ textArray: String array
+ file
loginValidation : Message
- type: login
- status: <valid> if valid
- text1: username
- text2: password
generateShare : Message
- type: share
- status: shared / failure
- text1: username
- text2: UUID of item
- textArray: array of users to share with
generateDeleteItem : Message
- type: delete
- status: deleted
- text1: username
- text2: UUID of the item
sendFileToClient : Message
- type: file
- status: fileMsg
- text1: username
- text2: UUID of the file
- file: file / folder
newUserValidation : Message
- type: signup
- status: <valid> if valid
- text1: username
- text2: password
- text3: email
accountSettingsResponse : Message
- type: settings
- status: success/failure
- text1: username
- text2: password
- text3: email
uploadValidation : Message
- type: upload
- status: valid / invalid
- text1: username
- text2: new UUID for the file
sendMainDirToClient : Message
- type: mainDir
- status: dirMsg
- text1: username
- files