TagIt.FM

High Level Design (HLD) Document

MM

Members: Jeremiah Thomas

Anna Malmberg

Sandhya Joshi

Zachary Wawrzaszek

Stavros Bannoura

**Revisions**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Primary**  **Author(s)** | **Description of Version** | **Date Completed** |
| **1.1** |  |  |  |
| **1.2** |  |  |  |

**Review and Approval**

|  |  |  |  |
| --- | --- | --- | --- |
| **Approving Party** | **Version Approved** | **Signature** | **Date** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**High Level Design Document Review History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Reviewer** | **Version Reviewed** | **Signature** | **Date** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1. Introduction 4](#_Toc44232062)

[1.1. Purpose 4](#_Toc44232063)

[1.2. Scope of this Document 4](#_Toc44232064)

[1.3. Definitions, Acronyms and Abbreviations 4](#_Toc44232065)

[2. Design Summary 5](#_Toc44232066)

[2.1 Overview of Design 5](#_Toc44232067)

[2.2 Design Issues 6](#_Toc44232068)

[2.3 Tools 6](#_Toc44232069)

[2.4 Libraries 7](#_Toc44232070)

[3. Design Details 7](#_Toc44232071)

[3.1 Security 7](#_Toc44232072)

[3.2 Hardware 7](#_Toc44232073)

[3.3 User Interfaces 7](#_Toc44232074)

[3.3.1. TagIt.FM Concept UI Diagram 8](#_Toc44232075)

[3.4 Internal Interfaces 8](#_Toc44232076)

[3.5 External Interfaces 8](#_Toc44232077)

[3.6 Architecture 8](#_Toc44232078)

[3.6.1 Software Architecture Diagram 9](#_Toc44232079)

[3.7 Reports 9](#_Toc44232080)

[3.8 Other Output 9](#_Toc44232081)

[3.9 Databases 9](#_Toc44232082)

[3.10 Top-Level Classes (Class Hierarchy) 9](#_Toc44232083)

[3.10.1 Client Main Class 9](#_Toc44232084)

[3.10.2 Server Main Class 9](#_Toc44232085)

[3.10.3 GUI Class 9](#_Toc44232086)

[3.10.4 User Class 10](#_Toc44232087)

[3.10.5 Tags Class 10](#_Toc44232088)

[3.10.6 Server Class 10](#_Toc44232089)

[3.10.7 DBhelper Class 10](#_Toc44232090)

[3.10.8 History Class 10](#_Toc44232091)

[3.10.9 FileManagerHelper Class 10](#_Toc44232092)

[3.10.10 HLD Diagram 10](#_Toc44232093)

[3.10.11 User Authentication Diagram 11](#_Toc44232094)

[3.11 Configuration Data 12](#_Toc44232095)

[3.12 Data Flows and States 12](#_Toc44232096)

[3.12.1 Data Flow Diagram 1 12](#_Toc44232097)

[3.12.2 Data Flow Diagram 2 13](#_Toc44232098)

# Introduction

## Purpose

The purpose of this project is to improve and speed up the process of searching for files on your computer. TagIt.FM is created to assist anyone who has ever had a difficult time locating a specific file within their file manager when they need it urgently. The client, MHC, is an insurance company and they need the application to be able to quickly locate important medical and financial documents of varying filetype. Customer service representatives especially will benefit from the application because it will both simplify the search process and reduce the amount of time they need to keep a customer on hold. Therefore, improving the mood of the customer and decreasing frustrations for the employee. The purpose of this document is to provide an overview of what will be included in the initial release of the application. The details of the project will be further discussed and developed in the Low-Level Design document.

## Scope of this Document

This document outlines the basic structure of TagIt.FM's interfaces and provides general information about the inner workings of the application. This includes descriptions and diagrams of how the GUI will communicate with the database for the creation, retrieval, editing and deletion of tags, as well as how the software will interface with the basic file manager on individual computers. Details about the design process including issues, tradeoffs, development and collaboration tools used are also included. Any libraries that were added to Python to assist with the construction of the software, are listed in the libraries section. Additionally, details are provided regarding security, hardware, reports and output, database, class structure, and flow of data.

## Definitions, Acronyms and Abbreviations

\*\*API\*\*: Application programming interface

\*\*ACID\*\*: Atomicity, consistency, isolation, durability

\*\*CLI\*\*: Command line interface

\*\*Class\*: A user-defined blueprint or prototype from which all objects are created

\*\*DBMS\*\*: Database management system

\*\*Git\*\*: Distributed version-control system used for tracking source code during software development

\*\*GitHub\*\*: Company that provides hosting for software development version control using Git.

\*\*GUI\*\*: Graphical user interface

\*\*HIPAA\*\*: Health insurance portability and accountability act of 1996

\*\*HTTPS\*\*: Hypertext transfer protocol secure is an extension of the hypertext transfer protocol that is used for secure communication over a computer network

\*\*IDE\*\*: An integrated development environment, such as PyCharm

\*\*Library\*\*: Reusable code, including functions, that can be used for common programming tasks

\*\*LAN\*\*: Local area network

\*\*Object\*\*: An abstract data type that contain multiple properties and methods, or even other objects as defined by its class

\*\*Python\*\*: An interpreted, high-level, general-purpose programming language

\*\*PyCharm\*\*: An IDE used specifically for the Python programming language

\*\*RDBMS\*\*: Relational database management system

\*\*SDLC\*\*: Software development life cycle

\*\*Slack\*\*: Proprietary business communication platform that features chat and file sharing

\*\*SQL\*\*: Structured query language: a language used in programming that is designed for data held in a relational database management system

\*\*SQLite\*\*: A database that operates directly from library function calls, without a separate process or server

\*\*Tag\*\*: A string without spaces which provides categorical information about its object

\*\*UML\*\*: Unified modeling language

\*\*UEWSG\*\*: Uniform English writing style guide, a set of rules and styles used to codify, standardize, and simplify writing

# Design Summary

## Overview of Design

The design of this project will be based around two main classes, the client main class and the server main class. These two classes will manage the interactions between the GUI class, the user classes, and the database.

The client main class will handle interactions between the user inputs and how the GUI will react. The server main class will handle interactions between the DBhelper and server classes as well as interactions with the database itself to manage the flow of data from the GUI to the database. The GUI class will instantiate objects that allow the user to view information about their file hierarchy, tags, and their search history. The GUI class is also where the user class will interact with these objects to select files, add tags or remove tags from files, create new tags or delete existing tags, and search for files based upon their assigned tags.

To access the application the user will need to log in through the GUI with a username and password that will be stored in the database. There will be three different levels of users, based on subclasses, where the increased level of user indicates increased privileges and gives them access to more methods that they can execute.

The basic user subclass will have a username and password and will be able to add and remove tags from files, as well as, search for files based on the assigned tags. The next level of user subclass will inherit the previous methods and properties, and will, in addition, be able to create and delete tags. The highest level of user subclass will inherit the previous methods and properties, and will be able to create, delete, and promote or demote user access levels. The GUI will also communicate through an external interface with the local computer's file manager so that the file hierarchy can be viewed and navigated through within the GUI.

The DBhelper class will reformat incoming and outgoing data so that the data will interact correctly with both the database and the programming language. All data will be transferred through an HTTPS server for security purposes. The tag database server, and the HTTPS server will be controlled by the server class.

## Design Issues

The project originally included many more features that were slowly filtered out as it became clear that there would not be enough time to include all of them in the initial release of TagIt.FM.

One of the things that was originally planned to be included was a predictive search feature. This feature would allow the application to start searching for files based on what the user had partially typed. Including this feature would have taken extra research and effort and was scrapped to increase focus on the main functionality of the application.

Another feature that was intended to be included was the ability for the user to personalize a color scheme for the GUI and tags. Again, this feature was pushed from the initial release as it is only a quality of life improvement and has no effect on the features that are required for the application to function.

The final feature that was not included in an initial release was including a way for the application to save the currently selected file when the application stops running, whether intentionally or accidentally. This is a quality of life improvement that could save the user a minute every day and is something that may be included in a future update instead of in the first version.

## Tools

Slack version four is used as TEAM's primary communication tool. On Slack, group members post updates to their current assignments, ask questions that they may need help with, and discuss plans for future deadlines. Slack is also used to host bi-weekly meetings, which are scheduled and documented on GitHub.

TEAM uses Slack for collaboration and discussion of critical areas of the project. Dedicated channels are set up for discussion, including an LLD channel, an HLD, an SRS channel and a meetings channel. Each channel is intended to primarily discuss matters related to the associated deliverable. Group discussions are held on Slack, and the group collaborate to find solutions to problems as they arise.

The other collaboration tool that TEAM is using, is a popular version control system named GitHub. It is used to make updates to files, merge and upload files based on the project requirements. Narratives are saved on GitHub in plain text using the extension of .txt and following the UEWSG format. Diagrams and other documents are saved in a format suitable for their purpose.

Each group member has posted their preferred work hours in the availability.txt document on GitHub, based on which meeting times of 8 PM every Tuesday and Thursday were decided. The members of TEAM have tasks and other assignments that are listed in the master branch in the readme document that are added by the project manager.

TEAM will be using PyCharm version 2020.1.2 with the build number 201.7846.77 as its IDE, this is a tool developed by JetBrains for use with the Python programming language. The responsibilities of the IDE are to integrate with SQLite, which is the group's chosen database language. The IDE will be used to develop the application when the SDLC is at the development phase and to implement testing procedures within that tool.

## Libraries

# Design Details

## Security

MHC deals with documents and records that must remain confidential due to HIPAA regulations. All data transfers will be handled using an HTTPS server. HTTPS will encrypt the data and stops potential hackers from being able to understand any data that they were able to intercept during a transfer to the database. The company network is private and only connected to the internet through a firewall, which forces hackers to try to find direct access the network. The physical security of the building will provide further protection and deny unauthorized personnel direct access to the network or server.

## Hardware

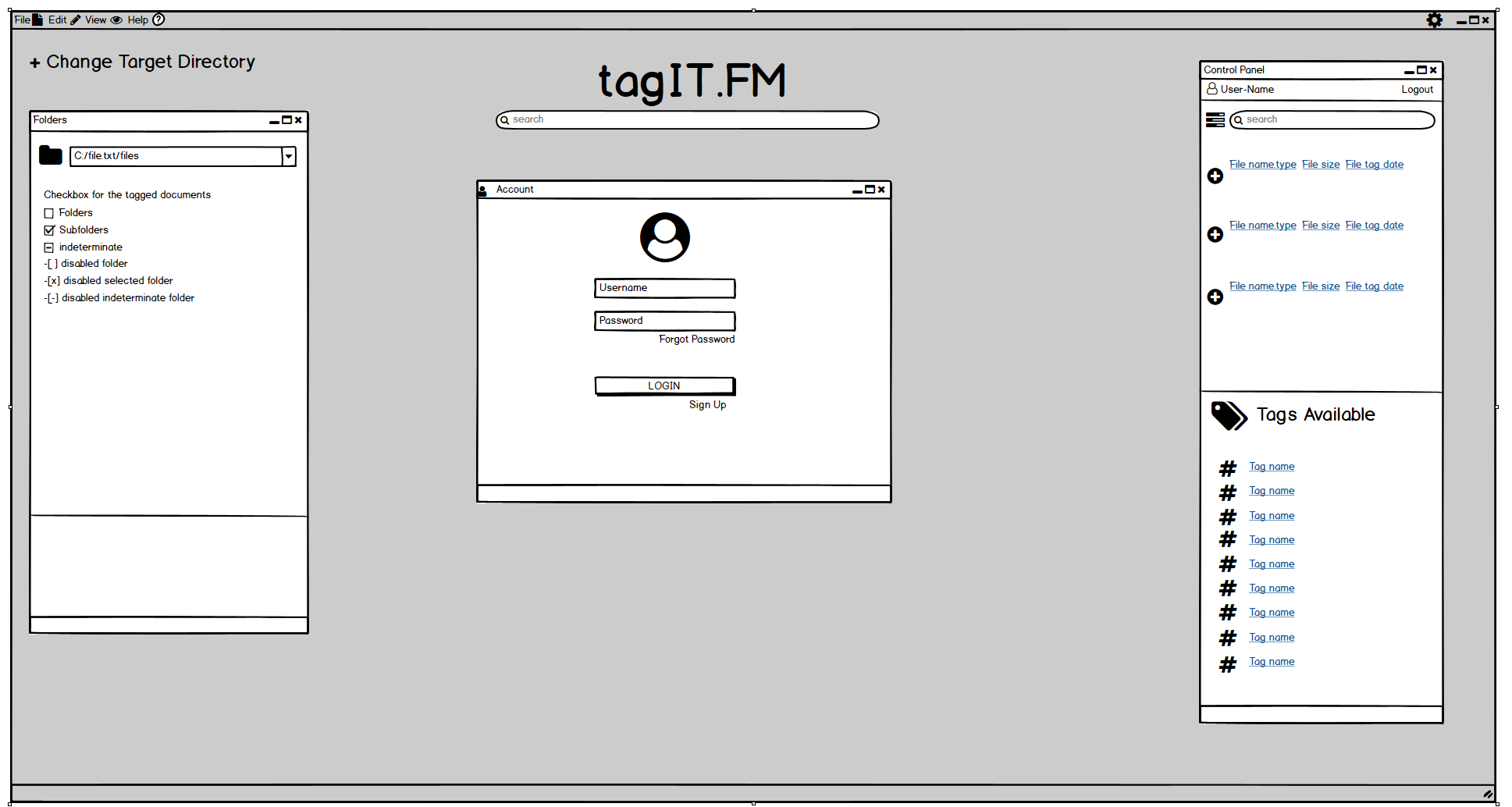
Required hardware for this project includes the following: a server for the tag database to be hosted on, a private LAN to facilitate communication between the employees' computers, which will host the GUI, and the database. Cables and routers will also be necessary to connect the tag database server to the LAN. The entire LAN should be hardwired for increased security. The TagIt.FM application will be able to run on any computer or laptop that is provided for the employees that is connected to the LAN.

## User Interfaces

The GUI of TagIt.FM will initialize as a window in the middle of the screen and can be resized and dragged to where the user wants it. A login screen will appear when the application starts, and the employee must enter their username and password to log in the application.

After a successful login, the GUI will display three separated areas. One side will be a panel where the employee can navigate their file tree and select a folder or file that they want to add or remove tags from. On the other side of the screen, there will be a panel that has all of the available tags that can be used, as well as a search bar to search for a specific tag that the employee wants to use. If the employee has the correct permissions, this is also where they will be able to create new tags and delete existing tags from the database. In the middle of the screen there will be a search bar where the employee will be able to search for files by entering the associated tags, and this is also where the returned files will be displayed.

### 3.3.1. TagIt.FM Concept UI Diagram



## Internal Interfaces

There will be a few internal interfaces at work in TagIt.FM due to several classes needing to pass data back and forth to each other. The GUI will be interfacing with the tags class to retrieve the list of created tags and add them to the display for the user to see. The GUI will also interface with the history class which will let the GUI to retrieve a list of the user's recent actions and display them to the user. The main class will be interfacing with both the user and GUI class to manage their interactions through the user's choices and actions in the GUI.

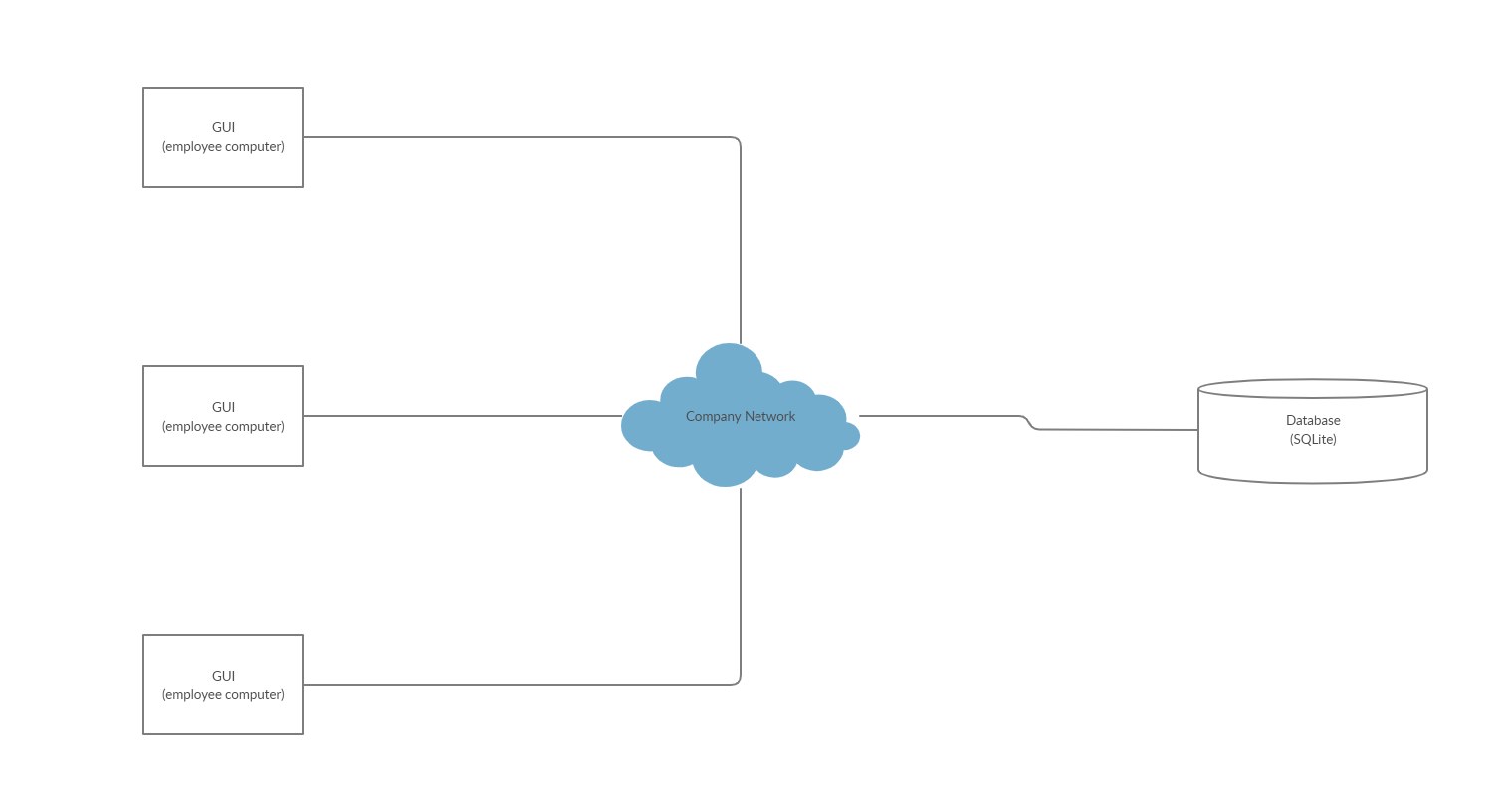
## External Interfaces

There will be two external interfaces involved the TagIt.FM. The first external interface will be needed to communicate with the computer's built-in file manager and pull the file hierarchy for the local computer into the application. The file hierarchy will maintain full functionality and navigational ability while being displayed within the GUI. The second external interface will be for communicating with the SQLite tag database. It will be responsible for reformatting transfer requests on both ends so that the database and the application can understand and utilize the information.

## Architecture

The architecture that will be used for this project is a client/server model. The client will have many different instances of the TagIt.FM GUI open on different employee computers. All these separate instances will be simultaneously communicating over the established company LAN and routed into the database using an HTTPS server. This architecture will support constant updates of the GUI based on any changes made to the data on the server side, when created by a different employee and instance of the program. This ensures that all the information remains constant and reliable for the employees who need to use it.

### Software Architecture Diagram



## Reports

## Other Output

## Databases

This project only requires the use of one RDBMS for the storage and management of user identification parameters, group permissions, filename and tag relations, and user histories. SQLite works well for MHC because it is a robust and lightweight system that boasts speedy query times which will be important for the customer service representatives who are trying to search for files in the most efficient manner possible. SQLite also offers high portability and will work across a wide variety of operating systems allowing the company to utilize any equipment they already have available. One final benefit of SQLite is that content can be updated continuously and automatically so that in the event of a power outage or crash little to no data will be lost.

## Top-Level Classes (Class Hierarchy)

### Client Main Class

This class will handle the classes for the client side of the application. It will call the GUI class and handle all interaction between the user inputs and the GUI. This will be a separate class from the server main class.

### Server Main Class

This class will handle the server and DBhelper side classes. It will handle the data for authentication purposes of the client’s requests.

### GUI Class

This will be the class that contains and displays the objects for the user interface.

### User Class

This class will contain usernames and passwords and will let the employee interact with the GUI. This class will also have two subclasses representing the level of the employee.

### Tags Class

This class will be responsible for holding all the created tags and displaying them for the employee.

### Server Class

This class will be used for setting up our server and defining the requests for sending and retrieving data.

### DBhelper Class

This class will be used for reformatting the data from Python to SQL, enabling the data to correctly interact with the database tables

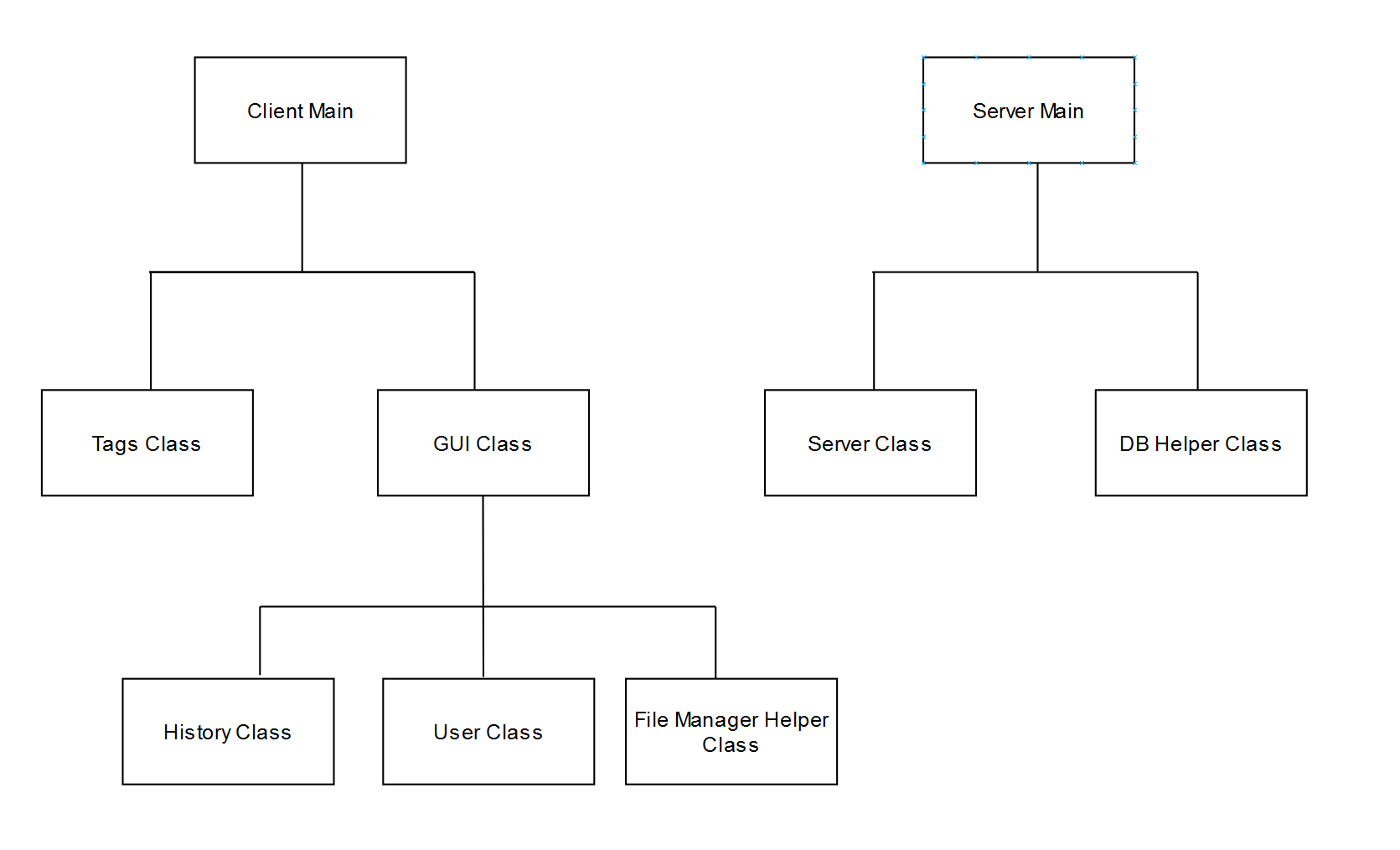
### History Class

This class will hold a list of the user’s recent actions, such as creating a tag, adding a tag to a file, or searching for a file by its tags.

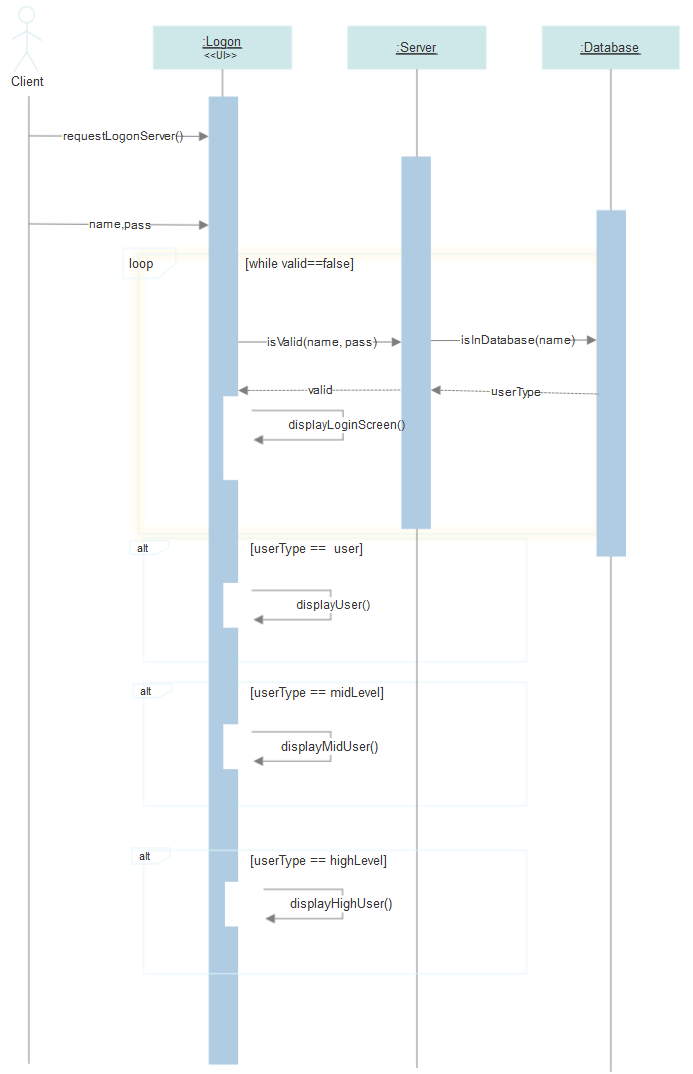
### FileManagerHelper Class

This class will be responsible for retrieving the computers file hierarchy from the file manager for display in the GUI.

### HLD Diagram



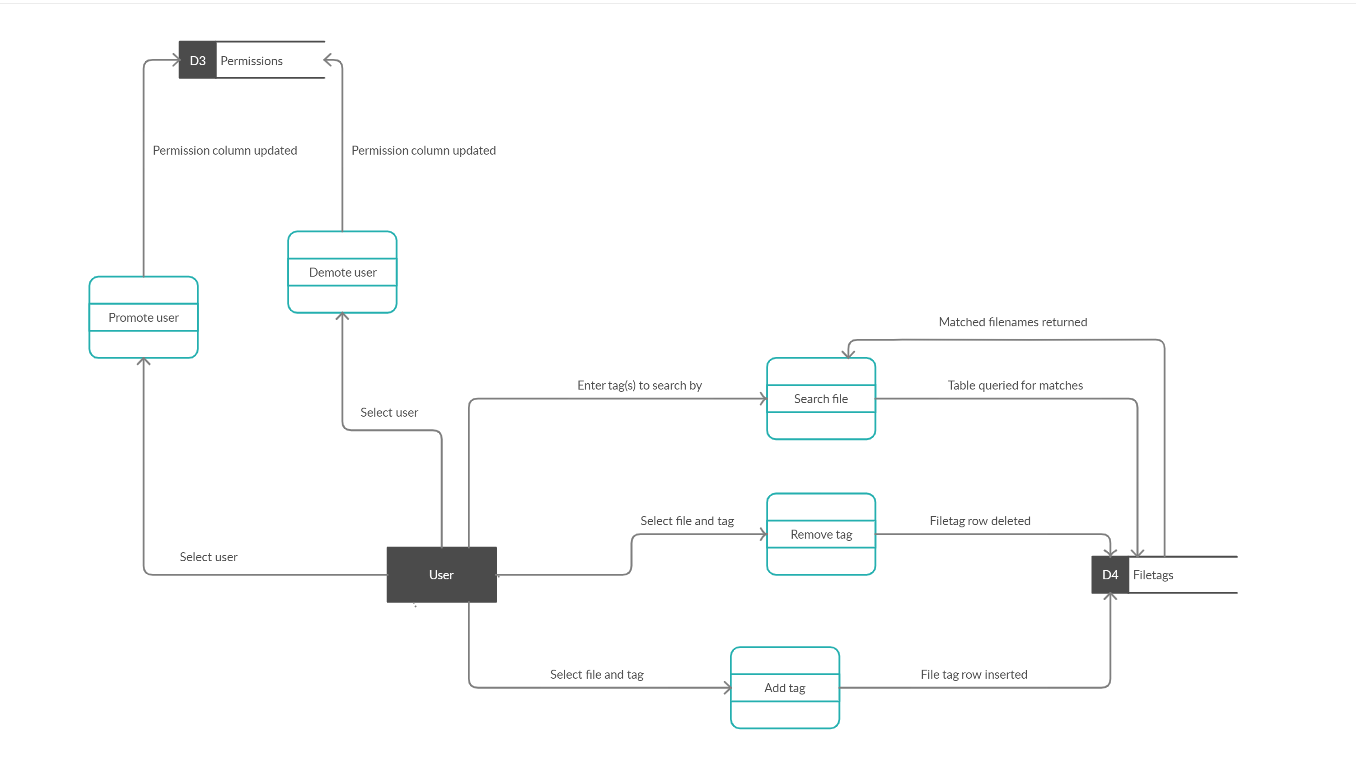
### User Authentication Diagram



## Configuration Data

## Data Flows and States

### Data Flow Diagram 1



### Data Flow Diagram 2

