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** | Zachary Wawrzaszek, et al | Draft Release | 2020-07-01 |
| **2** | Zachary Wawrzaszek, et al | Final Release | 2020-07-20 |

**Review And Approval**

|  |  |  |  |
| --- | --- | --- | --- |
| **Approving Party** | **Version Approved** | **Signature** | **Date** |
| Document Specialist | 1 | Anna Malmberg | 2020-06-30 |
| Lead Software Developer | 1 | Zachary Wawrzaszek | 2020-07-01 |
| Project Manager | 1 | Jeremiah Thomas | 2020-07-01 |
| Document Specialist | 2 | Anna Malmberg | 2020-07-19 |
| Lead Tester | 2 |  |  |
|  |  |  |  |
|  |  |  |  |

**High-Level Design Document Review History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Reviewer** | **Version Reviewed** | **Signature** | **Date** |
| Document Specialist | 2 | Anna Malmberg | 2020-07-19 |
| Lead Tester | 2 | Stavros Bannoura | 2020-07-19 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table Of Contents

[Table Of Contents 3](#_Toc45959215)

[1. Introduction 5](#_Toc45959216)

[1.1. Purpose 5](#_Toc45959217)

[1.2. Scope Of This Document 5](#_Toc45959218)

1.3. Definitions, Acronyms, And Abbreviations 5

[2. Design Summary 6](#_Toc45959220)

[2.1. Overview Of Design 6](#_Toc45959221)

[2.2. Design Issues 7](#_Toc45959222)

[2.3. Tools 7](#_Toc45959223)

[2.4. Libraries 8](#_Toc45959224)

[3. Design Details 8](#_Toc45959225)

[3.1. Security 8](#_Toc45959226)

[3.2. Hardware 9](#_Toc45959227)

[3.3. User Interfaces 9](#_Toc45959228)

[3.3.1. TagIt.FM Concept UI Diagram 10](#_Toc45959229)

[3.4. Internal Interfaces 10](#_Toc45959230)

[3.5. External Interfaces 11](#_Toc45959231)

[3.6. Architecture 11](#_Toc45959232)

[3.6.1. Software Architecture Diagram 11](#_Toc45959233)

[3.7. Reports 11](#_Toc45959234)

[3.8. Other Output 12](#_Toc45959235)

[3.9. Databases 12](#_Toc45959236)

[3.10. Top-Level Classes (Class Hierarchy) 12](#_Toc45959237)

[3.10.1. Main Class 12](#_Toc45959238)

[3.10.2. Server\_main Class 12](#_Toc45959239)

[3.10.3. Gui Class 13](#_Toc45959240)

[3.10.4. User Class 13](#_Toc45959241)

[3.10.5. Tags Class 13](#_Toc45959242)

[3.10.6. Https\_server Class 14](#_Toc45959243)

[3.10.7. Db\_helper Class 14](#_Toc45959244)

[3.10.8. History Class 14](#_Toc45959245)

[3.10.9. File\_manager\_helper Class 14](#_Toc45959246)

[3.10.10. Https\_client Class 14](#_Toc45959247)

[3.10.11. HLD Diagram 15](#_Toc45959248)

[3.10.12. User Authentication Diagram 16](#_Toc45959249)

[3.11. Configuration Data 17](#_Toc45959250)

[3.12. Data Flows And States 17](#_Toc45959251)

[3.12.1. Data Flow Diagram 1 17](#_Toc45959252)

[3.12.2. Data Flow Diagram 2 18](#_Toc45959253)

# Introduction

## Purpose

The purpose of this project is to provide an application that will improve the speed and accuracy of searching for files on a computer. TagIt.FM is designed to assist anyone who has ever had a difficult time locating a specific file within their file manager when they need it urgently. The app will be a reliable resource for any employee that is looking for a software solution that will assist in reducing the time it takes to locate files.

The client, MHC, is an insurance company and they need the application 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 that they need to keep a customer on hold. This, in turn, can improve the mood of the customer and decrease frustrations for the employee. This document will provide an overview of the design for 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 specifications of how the GUI will communicate user requests to the database for the creation, retrieval, editing, and deletion of tags. It also discusses how the software will interface with the basic file manager on individual computers and with their operating systems. Details about the design process including issues, tradeoffs, and development and collaboration tools used for the project are also included. Any Python libraries that will be needed to assist with the construction of the software are listed in the libraries section. Additionally, this document will discuss various details the application, such as architecture, security, hardware, reports and output, database, class structure, and flow of data, which are all located in chapter 3 .

## 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

**\*\*Hash\*\***: A scrambled representation of a password

\*\*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

**\*\*OOP\*\***: Object oriented programming

\*\*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

**\*\*Salt\*\***: In cryptography, a salt is random data that is used to safeguard passwords in storage

\*\*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 2 main classes, the Main class, and the Server\_main class. The Main class will manage the interactions between the Gui class, the User class, and the filesystem. The Server\_main class will oversee the handling of the Https\_server class and the database.

The Main class will handle interactions between user input and the GUI, directing how the GUI reacts to user requests. The Server\_main class will handle interactions between the Https\_server and Db\_helper classes, both of which will help manage interactions with the database and control the flow of data to the GUI. The Gui class will instantiate objects that allow the user to view information about the filesystem, tags, and search history. The User class is completely independent from the other classes and is where the user will interact with these object instances to add or remove tags from files, create new tags or delete existing tags, and search for files based on their assigned tags. The GUI will also be able to display reports on various application or user metrics.

To access the application, the user will need to log in through the GUI with their assigned username that will be stored in the database; this will occur through the Http\_client class. The password will be stored in the database using only salt and hash for security purposes. There will be 3 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 User superclass will have username attribute and will be able to add and remove tags from files, as well as search for files based on the assigned tags; most of these actions will take place via method calls to the Tags and History classes. The next level of user subclass is named Mid\_user. The Mid\_user class will inherit the previous methods and properties and will, in addition, be able to create and delete tags. The highest level of the User class will inherit from the Mid\_user class and add new methods to create and delete users and promote or demote user access levels. Every user class will have an id assigned to them, which will be used to confirm availability of permissions for executing CURD actions. 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 Db\_helper class will reformat incoming and outgoing data so that the data will interact correctly with both the database and the programming language. The class also will handle the creation of database queries in an OOP language. All data transfer will occur through an HTTPS server for security purposes. Https\_server class will control the tag database server and the HTTPS server.

## Design Issues

The project originally included several additional 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 have allowed the application to start searching for files based on what the user had partially typed out in the search bar. Including this feature would have taken extra research and effort and it 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.

One final feature that was excluded from the initial release was including a way for the application to save the currently selected file if the application stops running, whether intentionally or accidentally. This is a quality of life improvement that could perhaps 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 4 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 all the critical areas of the project. Dedicated channels are set up for discussion, including an LLD channel, an HLD channel, an SRS channel, and a meetings channel. Each channel is intended to primarily discuss matters related to the associated deliverable. General group discussions are also held on Slack, and the group collaborates to find solutions to problems as they arise.

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

At the start of the project, each group member 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. There are other alternatives like using Notepad++ which is an example of a text editor that the TEAM will also be using. 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

**\*\*http.client\*\***: Defines classes which implement the client side of the HTTP and HTTPS protocols.

**\*\*ssl\*\***: Provides access to transport layer security encryption and peer authentication facilities for network sockets, both client-side and server-side.

**\*\*os\*\***: Provides a portable way of using operating system dependent functionality.

**\*\*tkinter\*\***: Provides a robust, platform-independent windowing toolkit with text and graphics support.

**\*\*sys\*\***: Provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. The sys.path list is a commonly used feature of sys.

**\*\*timeit\*\***: Provides a simple way to time small bits of Python code.

# Design Details

## Security

MHC deals with documents and records that must remain confidential due to HIPAA regulations. Therefore, all data transfers within TagIt.FM will be handled using an HTTPS server. The HTTPS server will encrypt the data and stop potential intruders from being able to understand any data that they may have intercepted during a transfer to the database. The company maintains a private network and mandates the use of firewalls when connecting to the internet, which reduces the potential of DDOS attacks or hackers being able to find direct access to the servers. The physical security of the building will provide further protection and deny unauthorized personnel direct access to the network or server.

To provide another layer of security to TagIT.FM, MHC employees will be provided a username and password. The user account permissions will be based off their job-related responsibilities in the company. Since every user will have the ability to complete tasks in the application, TagIT.FM will have 3 different roles assigned to user accounts that will restrict the users' ability to modify and access critical data within the application. To create or delete tags within TagIt.FM, the user is required to log in with a Mid\_user level access. For administrative tasks, there is a High\_user level which can create or delete user accounts and grant accounts a different access level.

This layered approach in the application will provide a higher level of security that will offer the client a sense of comfort about how their data is handled. It will also organize the user account structure to prevent any confusion regarding who can handle the higher administrative tasks.

## Hardware

TagIt.FM requires a server that will host the database for handling tags and a private LAN to facilitate communication between the employee's computers which will host the TagIt.FM GUI; cables and routers will also be necessary to connect the tag database server to the LAN. The application will be able to run on any Windows or Linux computer or laptop that is provided for the employees and is connected to the LAN.

## User Interfaces

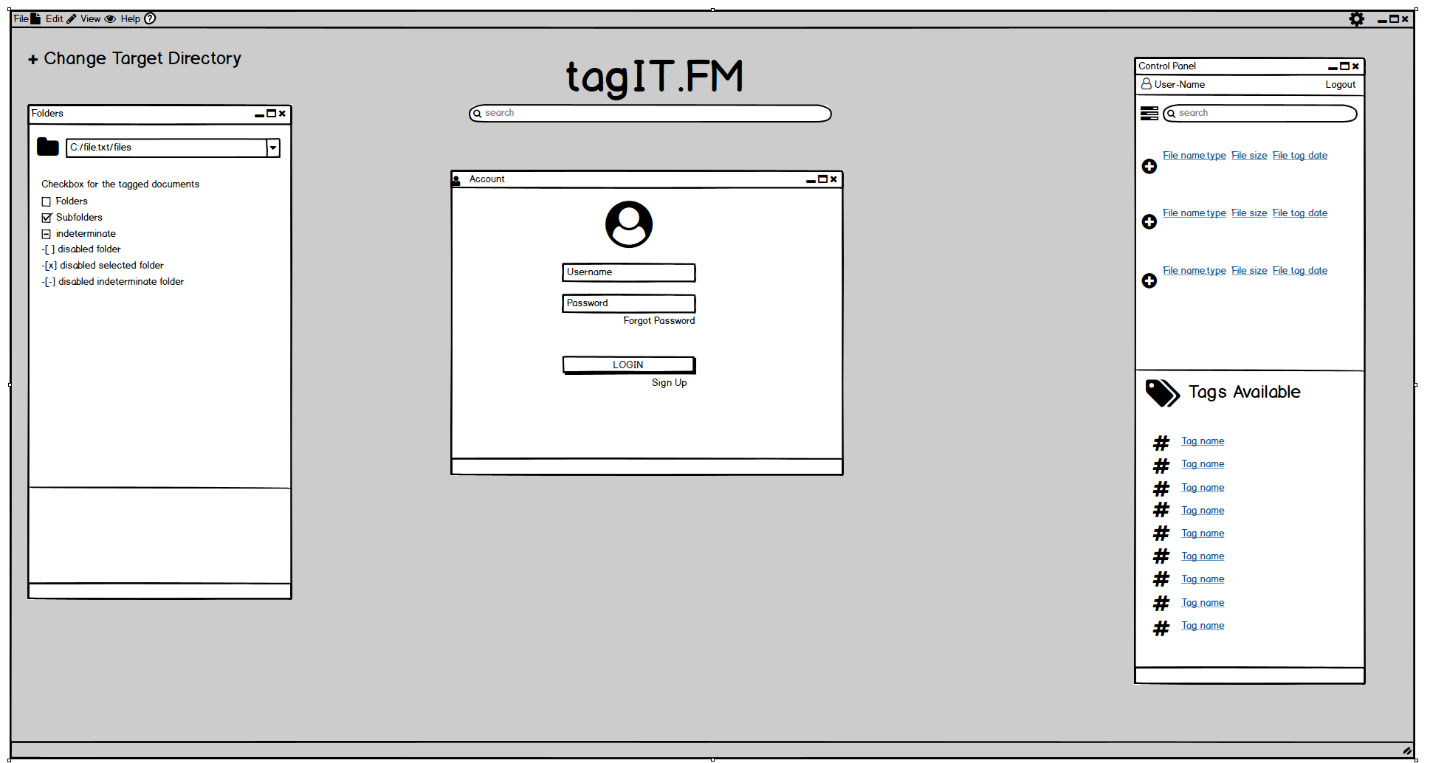
The GUI of TagIt.FM will initialize as a window and can be resized and dragged to where the user wants it. A login screen will appear when the application starts, and the employee must enter their username and password to log in to the application.

After a successful login, the GUI will display 3 separate 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 the available tags that can be used, as well as a search feature 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; this is also where the returned files will be displayed.

The GUI will also feature access to the reports outlined in section 3.7 . The current user's username will be displayed in the corner of the window so they are able to confirm they are logged in under the correct account. A logout button will allow the user to exit their session and bring them back to the login screen. In the file hierarchy area of the GUI, the user will also be able to observe the file creation date and current size of the file as outlined in section 3.8 .

### 3.3.1. TagIt.FM Concept UI Diagram

The following diagram illustrates the basic GUI layout that will be used in the TagIt.FM application. It features the 3 main panels, including the account login panel, the control panel with a search feature as well as a list of recently used tags, and the folders panel, which shows the currently selected directory. Additionally, there is the main search bar for finding files by their associated tags.



## 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 class 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 class will also interface with the History class which will let the Gui class 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 classes to manage their interactions through the user's choices and actions in the GUI. The Main class will also create an instance of the Https\_client class which will communicate data retrieved from the server side of the network through the Main class to the rest of the classes.

There will also be internal interfaces necessary on the server side of the network. The Server\_main class will create instances of both an Https\_server class and a Db\_helper class. The Db\_helper class will be responsible for the encapsulation of data that will be sent to the database, as well as the de-encapsulation of data retrieved from the database. The Https\_server class will handle the transfer of data to and from the Https\_client class. The data retrieved from the Main class will be used to update the database and the data sent back to the client will be used to update the Tags and History classes.

## External Interfaces

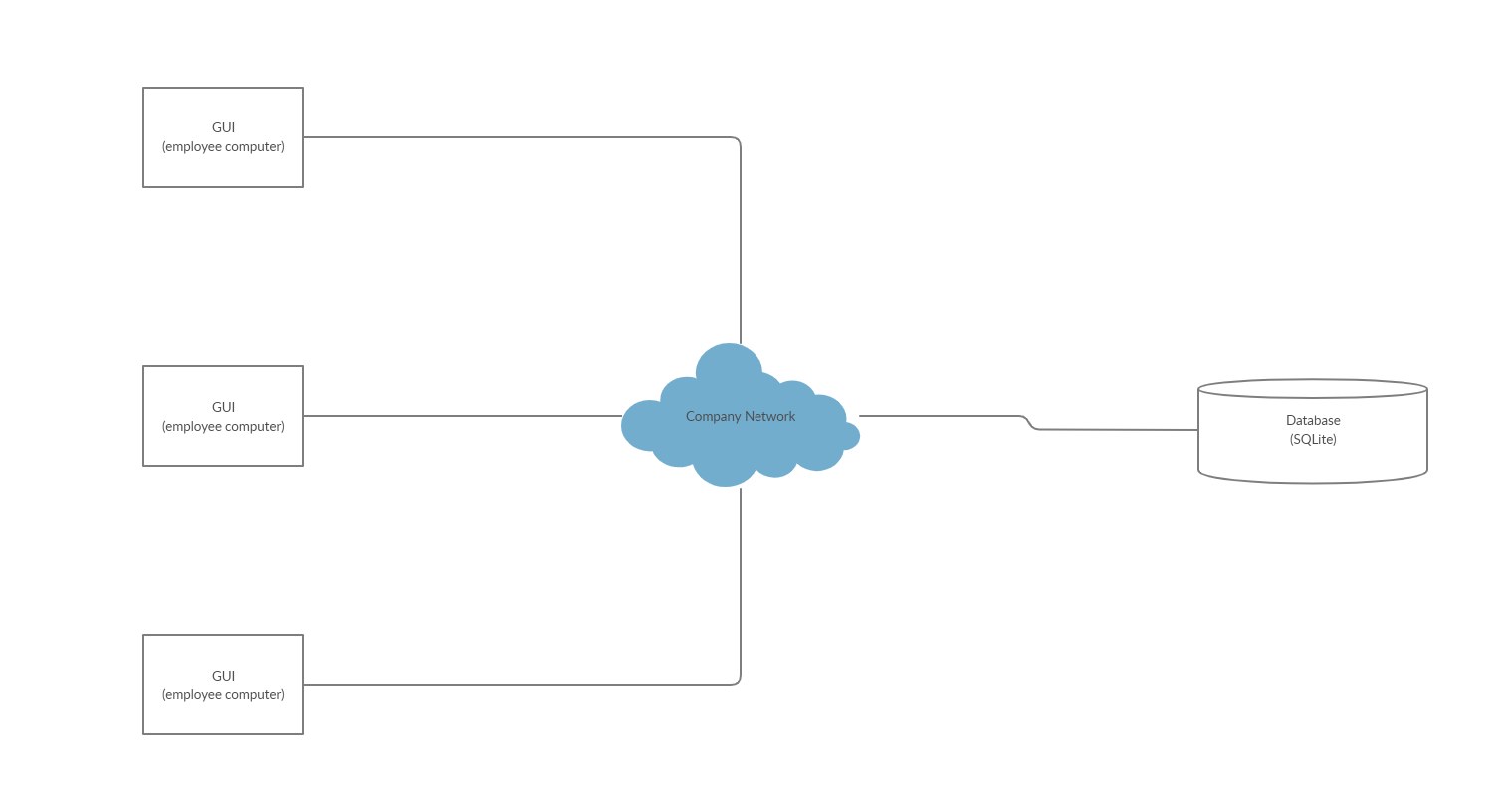
TagIt.FM will feature 2 external interfaces: 1 on the client side and 1 on the server side. On the client side, the application will interact with the local filesystem in order to provide an up-to-date listing of the local computer's files and directories; this will occur without any filesystem locking or other interference which would infringe upon normal filesystem operations. On the server side, the app will interact with an SQLite database, which will contain tag-related information and user accounts and permissions. This interface will be responsible for encapsulating all requests into discrete SQL statements and de-encapsulating all responses back into the standard datatypes that are recognized by the application.

## 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 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

This diagram illustrates the client/server model architecture that will be used for the TagIt.FM application. The clients are shown on the left, connecting through the LAN to the server on the right side of the diagram.



## Reports

Reports will be generated through pre-programmed SQL statements. In order to ensure confidentiality is maintained, the system will verify that the user who requests a report has the required access level of that report.

Two of the reports will require High\_user access to the application and are intended to use for auditing purposes. The first report will display the total number of users and the total number of tags available for use. The second report will display a database view that allows the user to view each user's current level of permissions.

The final report will allow a user to select a tag and it will display all the files that are associated with that tag that the user could normally access. This will occur without revealing any confidential information included in the displayed files.

## Other Output

The GUI will display some other information for the user's convenience: they will be able to view the last modified date and current size of the file selected and will see the username of the currently logged-in user.

## Databases

This project will only require the use of a single 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)

### Main Class

The Main class will handle all other classes on the client side of the application and well as the interaction between them; it will be a separate class from the Server\_main class. It will initially have 7 methods that it will use to structure the data for the MHC employees. The init method will initialize the GUI, the HTTPS Server; this method will be kept private while the rest of the methods will be public so that if an error occurs, or a user logs out and wants to log back in, they can be restarted individually.

The create\_gui method will primarily be involved in creating the application's GUI and then updating the associated private variables required for that function. The create\_https\_client and create\_user methods work in the same manner as the create\_gui function. These 3 methods will store variables that will be used in by the User superclass and subclasses to handle any tasks: for example, if the user would like to create a file tag in the GUI they can call the create\_gui method. The last 3 methods, read\_gui, read\_https\_client, and read\_user, will primarily be in charge of reading the memory locations of the objects so information can be exchanged with other objects.

### Server\_main Class

The Server\_main class will manage 2 other classes, namely the Https\_server and Db\_helper classes. The number of methods in this class are yet to be determined, but there will be an init function handle the instantiation of the Db\_helper and the Https\_server objects. The Https\_server and Db\_helper classes will also have init methods and respective private variables so that they can pass their location to other objects.

### Gui Class

The Gui class will contain and display the objects for the user interface. It will connect with the History class to ensure that the user has access to previously-searched tags. Additionally, the Gui class will communicate with the computer's file manager through the File\_manager\_helper class. This will allow for the retrieval of the computer's entire file hierarchy, which can be displayed to the user within the GUI so they can navigate through it and select the file(s) they want to tag or interact with. The Gui class will also contain a method that will allow the user to view the reports detailed in section 3.7 in a separate area. The Gui class will provide information about the creation date and current size of the file selected in the file hierarchy area of the window, as well as displaying the user's username next to the logout function so they can confirm they are logged in under the correct account.

### User Class

The basic User class will contain usernames and id attributes and will let the employee interact with the GUI. This class will have two subclasses representing the level of the employee, each subclass inheriting from the previous class and adding its own methods and attributes.

Each level of the user classes will be set up using this format due to the authentication requirements outlined in the SRS document. Depending on their job duties at MHC, user accounts will be assigned either as High\_user, Mid\_user, or User at creation but the level may be changed later by a High\_user. The level of the account will give the employee access to specific functions within the TagIt.FM application so that they will be able to complete the tasks required by their role.

The User class will primarily involve 7 different methods to create the structure for the remaining 2 subclasses; the methods are are init, login, create\_file\_tag, delete\_file\_tag, read\_file\_tags, read\_history, and read\_id. The init method will handle initializing the user object, which assigns the username and password while calling the login function. The login method will be responsible for handling the client's response to a login request with the given username and password. The create\_file\_tag, delete\_file\_tag, and read\_file\_tags methods will check that the user has the required user level prior to completing a task then quickly complete their tasks—creating, reading, or deleting a file tag—using the given parameters and arguments. The read\_history method will primarily be responsible for gathering the information from the user's history that stores recent tag searches and read\_id will return the user’s ID.

### Tags Class

The Tags class will be responsible for holding all the created tags and tag descriptions and displaying them for the employee. The class will receive constant updates from the database, aided by the Db\_helper class, and it will take into account any changes that have been made to the availability of specific tags, ensuring that both the database and the Tags class stay up to date.

### Https\_server Class

The Https\_server class will be used for setting up the server and defining the requests for sending and retrieving data. It will interface with the Https\_client class, which is where the requests for the data needed for the other classes will come from. It will be compiled using the ssl and http.client libraries, which allow for secure transfers using HTTPS protocols and certificate authentication.

### Db\_helper Class

The Db\_helper class will be used for reformatting the data from Python to SQL and vice versa, enabling the program to correctly transfer data to and from the database tables. The class will also facilitate the use of Python from within the application to create the SQL functions and statements needed to manage and manipulate the database more generally. Finally, it will have the responsibility of handling errors that may come across from the SQLite database due to dirty data or incorrect requests.

The database itself will be normalized according to 3NF standards. Because of this, it will feature fewer dependencies among the tables that are created and the tables will be organized to a level where there is no data duplication within them. For example, there will be no many-to-many relationships within the database tables, helping to ensure 3NF is maintained.

### History Class

The History class will hold a list of the user's recent searches to some arbitrary depth. This class will cooperate with the GUI as well as the database to connect the current user to up-to-date history.

### File\_manager\_helper Class

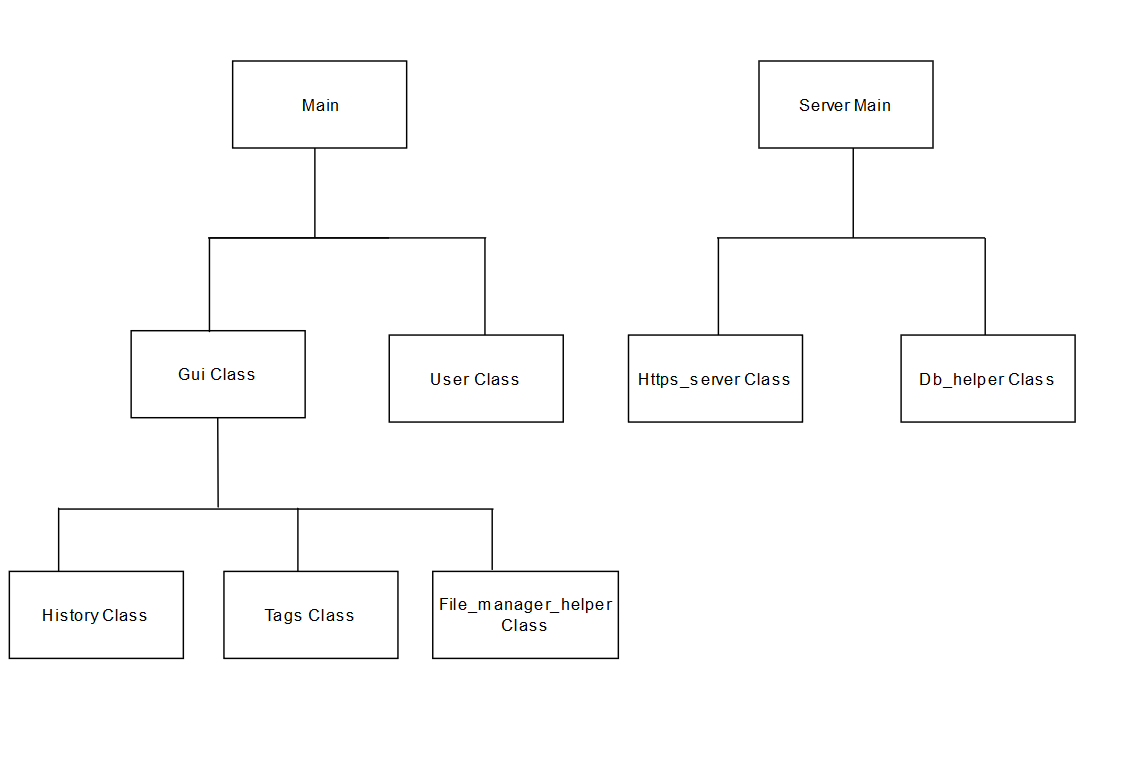
The File\_manager\_helper class will be responsible for retrieving the computer's file hierarchy from the file manager for display in the GUI. It will be called by the GUI when a user searches for a tag; the subdirectory checkbox in the GUI will be read in order to determine if subdirectories will be searched. The class will then interface with the local filesystem and return a Python-friendly list of available files to the GUI, files which may potentially be tagged. It will also determine whether the client is using Windows or Linux and may need multiple methods to handle those differences.

### Https\_client Class

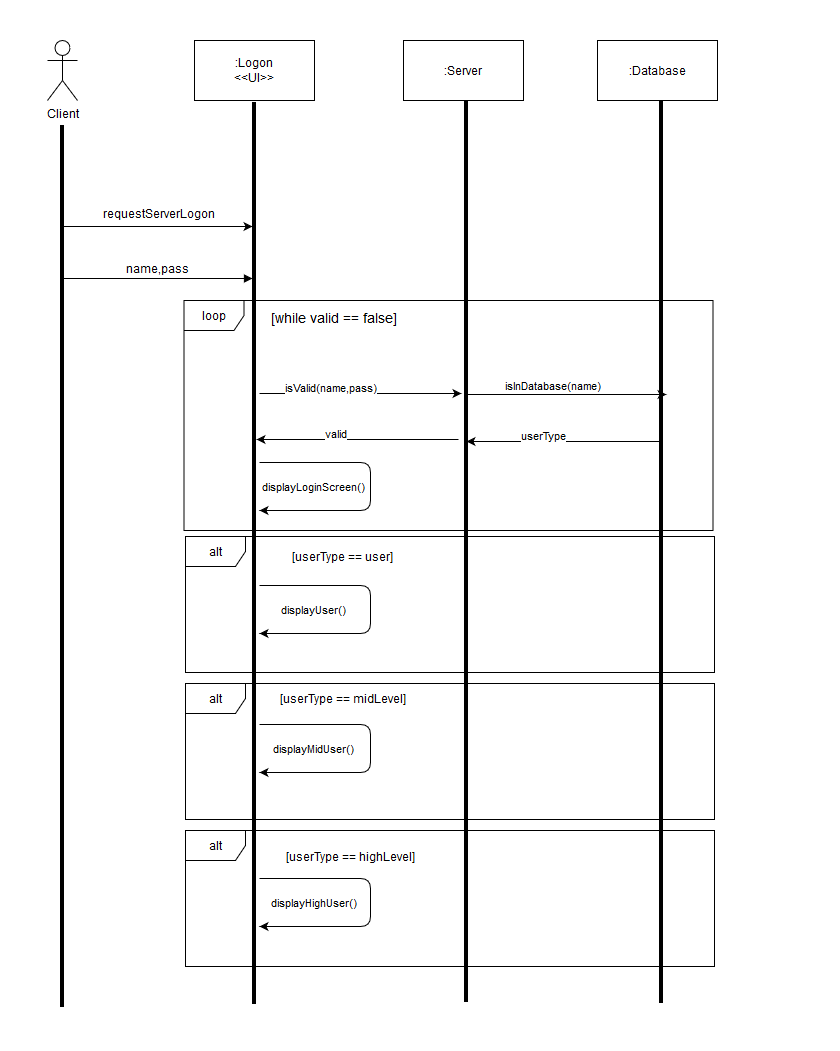
The Https\_client class handles the requests and transfers of data to and from the server side of the project architecture. It will utilize both the http.client and ssl libraries to provide secure transfers of data between the client and server. The http.client library will provide HTTPS support if the code is compiled using the ssl library. This is accomplished through the ssl library's methods that allow for certificate authentication and its ability to create secure sockets.

### HLD Diagram

The following diagram explains the classes that will implemented during the design phase of the project. It contains 2 main classes, Main and Server\_main. The diagram displays the Main class on the left as being responsible for handling the 5 classes in its hierarchy: Gui, User, History, Tags, File\_manager\_helper. On the right side, the Server\_main class is only responsible for 2 other classes: Https\_server and Db\_helper.



### User Authentication Diagram

The authentication diagram explains the process of a client logging in to TagIt.FM with proper credentials. The system validates the username and password information in the database and confirms the user level of the account, checking each level until it finds a match.

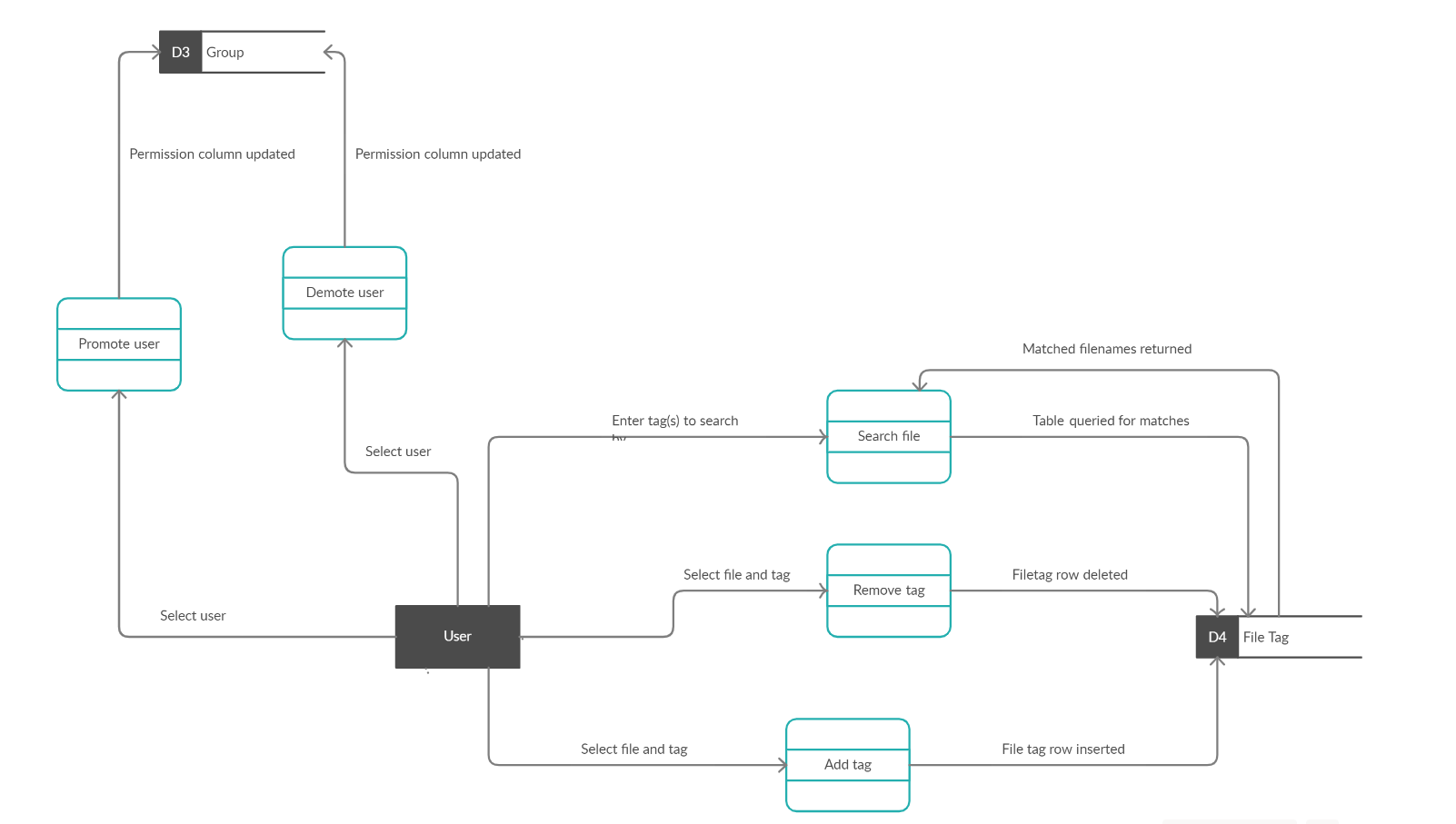
## Configuration Data

At its initial release, TagIT.FM will feature 2 important points of configuration: the initial user groups and the initial admin account. Since the User class and subclasses assume a certain levels of permissions, the database needs to be configured with those groups already in place and properly named. Secondly, in order to administer the application, an initial account with High\_user privileges will be set up on the server machine; only the first account on the server will have the privileges, preventing clients from accidentally or maliciously gaining system rights they should not have access to.

## Data Flows And States

### Data Flow Diagram 1

Data flow diagram 1 illustrates the functions that will be available to the High\_user class. This class will be able to change the level of users, in addition to the tag creation and deletion functions that it will inherit from the Mid\_user class, and the search function it will inherit from the User class.



### Data Flow Diagram 2

Data flow diagram 2, illustrates some of the functions that will be available to a user depending on their level, and how they will interact with the database tables. The login function as well as the create/delete user functions will interact with the User table. The create/delete tag functions will interact with the Tag table, and the update or clear history functions will interact with the History table.