**Assignment 3 – Centralized P2P File Sharing Application: FILESTORM**

**Server Implementation**

The system starts by running SocketServer.java, which starts up the TCP socket and the UDP socket. Both sockets then create new threads so that they can run alongside each other. The TCP socket creates a new thread using the SocketClientHandler.java thread, and the UDP socket creates a new thread using UDPServer.java, which starts a UDPServerThread,java thread.

The SocketClientHandler is used to separate the different connections that the TCP Clients can send to it. The messages that it replies to are:

* CREATE: This connection creates a new client profile and is called when the clients connect to the server for the first time. Is not called on subsequent connections.
* APPEND: Called alongside the CREATE connection when connecting for the first time, and is also called every other time that a client connects to the server. Looks at all of the files that are saved in a specified file path (C:/Users/Documents/FileStorm) and appends them to a list. The list can be searched using the SEARCH connection, and if it receives a file match it will download that file from the peer that it was found on if they are still connected to the server.
* SEARCH: Using the search function the Client GUI, users can type in what file they want to download. This tells the server to look through the master list for the file if it has the key has been uploaded and then tells the connecting Client which Peer has a copy of that file. It then downloads that file from them.

Files are appended and searched on the server using what we are calling the Master List. It is an ArrayList of String ArrayLists that holds all of the file information along with the IP and Port information so that the files can be downloaded. The UDP Datagram Socket is used to get pings from the clients, and will cancel their connections and free up the sockets if they have not pinged back to the server within a 30 second interval. All incoming connections are given their own threads.

**Client Implementation**

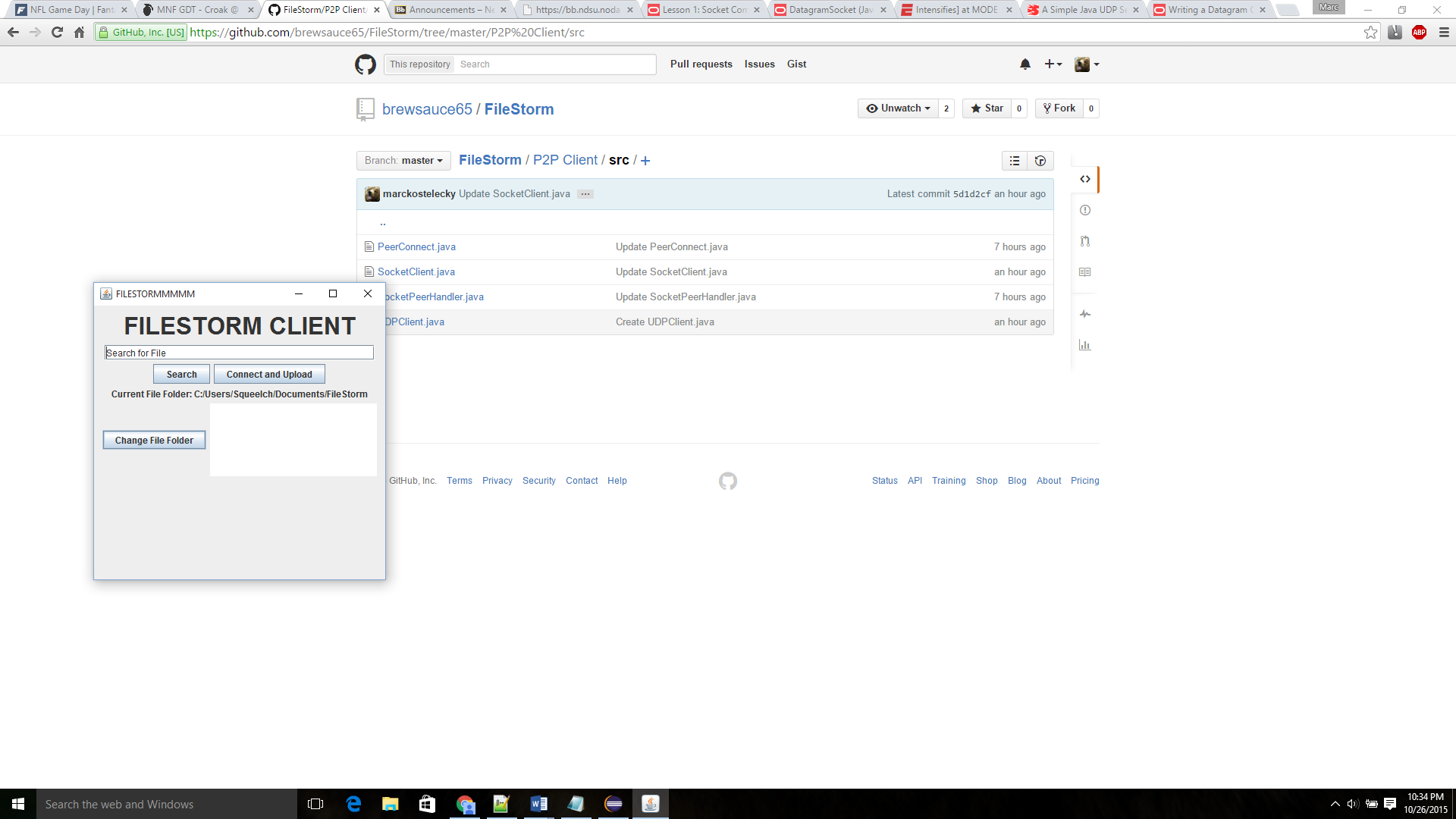
The Client program is run using SocketClient.java. This carries a Graphical User Interface that allows users to connect to the central server, upload files, search for files, and download them.

Figure : FILESTORM Client GUI

The GUI has a search field along with a **Search** button that send a SEARCH query to the server with what was typed into the search field. Searching will not work until you have pressed the **Connect and Upload** button, which connects you to the server and also uploads all of the file keys for the files in your Current File Folder which is listed underneath. There is a **Change File Folder** button to change the file folder, which allows you to change the file path to send and download files to and from. The field next to that is intended to show which files are being uploaded and downloaded, but we did not have time to finish that functionality and it is only shown in a Java Console at the moment.

On the client backend, once the user clicks on the **Connect and Upload** on the GUI the client creates three connections, one to the central server using TCP, one to the central server using UDP, and then it creates its own socket on a different port to accept incoming peers to download files from it. Once the server receives a TCP connection, the server appends all of the files that are in the clients file folder onto the master list and saves it with the clients IP Address and username. It will not append files that are already known, but will append new files that have either been downloaded from another user or if the user moves files to be uploaded there. The server sends the client a message with the files and the connection it received.

**Peer Implementation**

When a client has connected to the server, it also opens up its own connection for incoming peers. These are handled with the SocketPeerHandler.java and PeerConnect.java classes. SocketPeetHandler.java is how the server is told which clients have connected as peers, and is where the file keys are uploaded to the server. This is done with its own thread so that you can upload new files at the same time as downloading files form another peer.

PeerConnect.java works very similarly to the central server. It waits for connections and alerts when another client has connected to it for a file download. It creates a new SocketPeerHandler thread so that multiple users can download from it at once.