Assignment 5

SEG2105 B

Student 1: Samy Abidib (6909624)
Student 2: Sarmad Hashmi (7249729)

Problem Statement

Share a client's files with multiple other clients without saving the actual files on the server.

Requirements

We present a web application that will utilize a client-server-database architecture. In this system, the client can share selected files and folders as well as access the files that other clients have made available. Clients can register as users and add a profile image. Clients can see what other clients are connected according to a username they enter when they launch the web application. Clients can select who they would like to share their files with. There can be multiple files shared with multiple clients.

Information about the IP of currently connected clients and the folders they have available to stream are kept in the servers memory and are not stored in persistent memory on the server. However, information such as a client's username, encrypted password, and profile image are kept in the server's database. Throughout a client's session information about the folder's they have chosen to share are kept in persistent memory on the client side. Users can search through the available folders and files of other users.

Users can request a file by clicking a file from the available files of another user. Once a file is requested, the requesting user is called User A and the sending user is called User B. User A sends a message to the server requesting the file from User B. The server sends a message to User B requesting the file. User B initiates a stream of data that contains the file, the server receives this stream and forwards it to User A.

Depending on the file type, different things will occur on User A's client. If the file is an audio or video file and is playable by a browser, it will be streamed. Otherwise, the file will simply be downloaded.

Use Cases

1. **Share a folder or a file**

Actor Actions	System Responses
1. Choose “Share” command	2. Displays file dialog box
3. Specify files or folders to share	4. Display all files about to be shared
5. Confirm selection	

2. **Stop sharing a file or folder**

Actor Actions	System Responses
1. Select files or folders to remove	
2. Choose “Unshare” command	3. Display all files about to be unshared
4. Confirm selection	

3. **Add or remove users to share with**

Actor Actions	System Responses
1. Choose already shared files or a folder	
2. Choose “Share With” command	3a. Display all users stored in the database 3b. Display users that have already been shared these files or folders.
4. Specify a user or a list of users to start or stop sharing the file(s)/folder with	5. Display files about to be shared or unshared and with whom
6. Confirm selection	

4. **Download a file**

Actor Actions	System Responses
1. Select a file	2. Confirm if client has access to file
3. Choose “Download” command	4. Provide requested file to client
5. Download the file	

5. **Stream an audio or video file**

Actor Actions	**System Responses**
1. Select a file	2. Confirm if client has access to file
3. Choose “Stream” command	4a. Confirm if file is audio or video file 4b. Play the file in the browser

Architecture

The architecture of this system will be relatively complicated as it will be a real time system that contains many security issues.

On both the client and server, the main logic programming will be done in JavaScript. The client will utilize HTML and CSS for the UI, while JavaScript will be used for the logic. The client will also use JQuery which allows developers to make use of JavaScript without writing out too much syntax. On the other hand, the server will utilize JavaScript as well, because it uses a NodeJS server. The server will also use SQL to request information from the database, as well as store data on it.

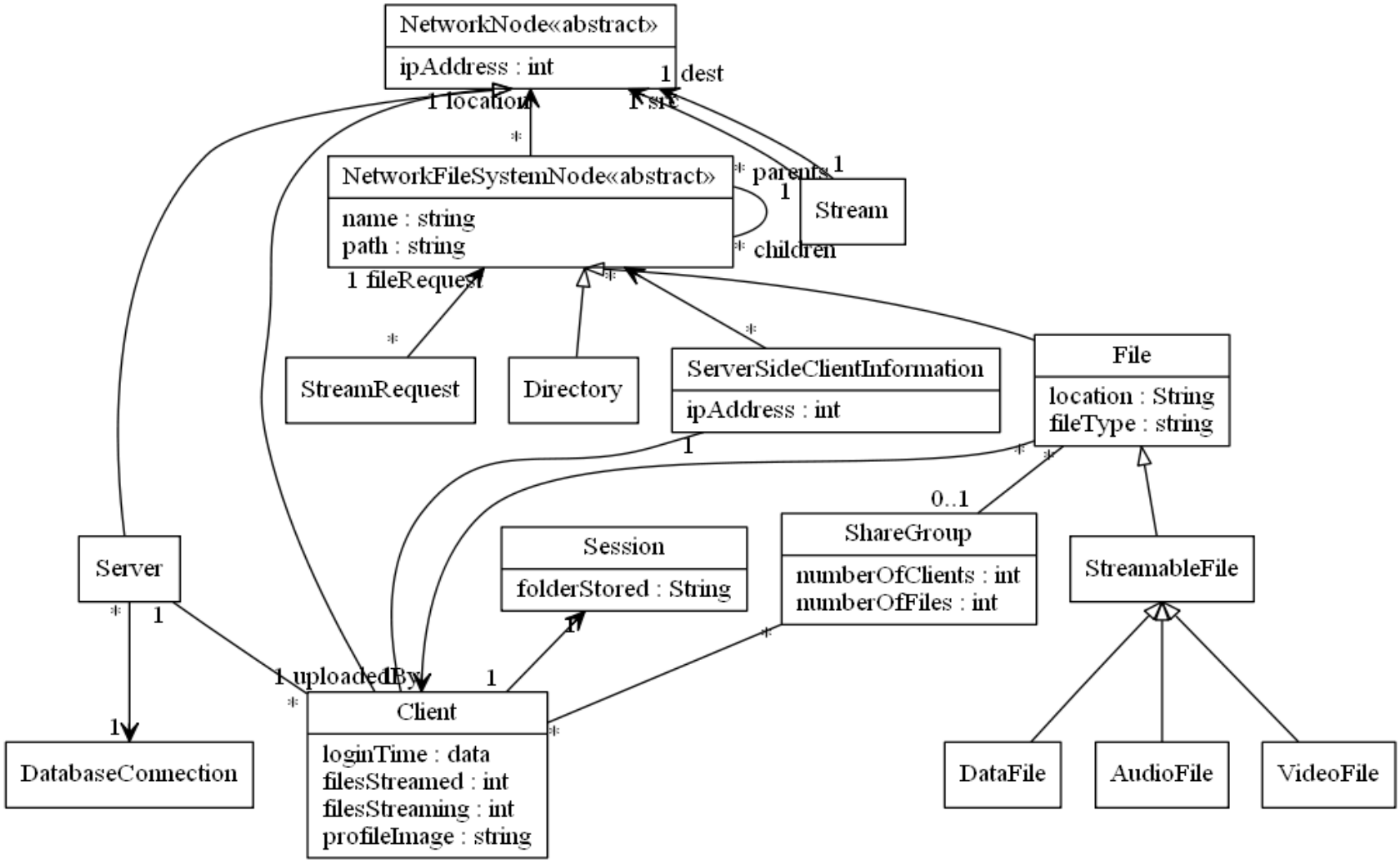
We implement the web application using an HTTP server. We implement this HTTP server using NodeJS and the Express web application framework. NodeJS is in a way, the client-server framework being used for this application. There is a NodeJS server running at all times, and whenever a client connects to the application, NodeJS creates a connection for them (called streams). We utilize NodeJS streams in order to create new client connections and provide clients with arbitrary data which can be messages or files. The use of streams allows streaming of music and videos between clients.

We extend the use of streams on both the server and client side by using Socket.IO and Binary.js. Socket.IO is a router for GET and POST messages sent to the server, it also implements an always on HTTP connection utilizing Web Sockets. Web Sockets allow messages to be passed between server and client without an HTTP header

Binary.js is an adapter for NodeJS streams that implements a bi-directional Web Socket that streams uncompressed binary data to and from clients. Binary.js allows the server to provide data or files to clients without actually storing them in a persistent database. Therefore, in this architecture, the server stores all the user information and files, and provides these files to clients whenever requested, thus creating interactions between a client and a server.

We will also be using GIT for version control. This will help us keep track of changes and allow us to work together on the application without conflicting each other's work.

UML Class Diagram



Client and Server Messages

Client to Server messages:

Client: connect
Client: login <username> <password>
Client: register <username> <password>
(The following messages can be sent to the Server after successful Client authentication)
Client: get listOfSharedFiles
Client: get listOfFilesSharedWithMe
Client: get listOfOnlineClients
Client: upload <list of files or folders>
Client: share <list of files or folders>
Client: shareWith <list of files to share> <list of clients to share with>
Client: unshare <list of files to unshare>
Client: unshareWith <list of files to unshare> <list of clients to unshare with>
Client: download <file>
Client: stream <audio or video file>

Server to Client messages:

Server: Connected.
Server: authenticate Client <username> <password>
if registering
Server: <username> is now registered.
Server: Successfully registered and authenticated.
If unsuccessful authentication:
Server: Invalid username and password, retry or register.
If successful:
Server: Successfully authenticated.
Server: <username> connected.
Server: <username> disconnected.
Server: <send list of files that client is currently sharing>
Server: <send list of files that other clients are currently sharing with this client>
Server: <retrieve all other online clients and send them to all clients>
Server: Provide file stored on <client1> to <client2>.
Server: <username> has shared <list of files> with <list of usernames>.
Server: <username> has uploaded a file: <filename>.
Server: <username> has downloaded a file: <filename>.
Server: <username> is currently streaming an <audio/video> file: <filename>.
Server: <username> has unshared <list of files>.
Server: <username> has unshared <list of files> with <list of usernames>.

Minimal Code

There is minimal working code provided for this assignment as well. The minimal code includes 3 classes, a server and client framework, as well as index.html. The functionality it achieves so far is the server allows clients to store files in the server's memory and stream audio/video files.

In order to run the provided code, please follow the instructions:

1) Extract the attached file "MinimalCode.zip" into any directory.
2) Install nodejs from http://nodejs.org/.
3) Open up a command line or terminal (if using Mac) as administrator.
4) Navigate to the directory you extracted MinimalCode.zip in, from the command line/terminal.
5) Enter "node index.js" without quotation marks.
6) The server will now be running at localhost:3000 (if the port isn't blocked on your computer).
7) Navigate to http://localhost:3000 with your browser.

In order to test the application, drag and drop any file onto the webpage where it says "Drag files here" (preferably something simple like an image or an mp3). If an audio/video file is dropped, it will begin playing the file. Otherwise, the file remain in the server's memory.

This uses the File, NetworkNodeStream and ShareGroup classes, which can be found in the **includes** directory. The Client and Server classes are provided by NodeJS.