

# **Assignment 7**

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.

We store the client and file information on the server memory. Throughout a client's session, information about the files they have chosen to share are also kept in server memory. Users can search through the available files of other users that have been shared with them.

Users can request a file by clicking a file from the list of files that are available to them for streaming or downloading. Once a file is requested, the requesting user is called User A (destination) and the sending user is called User B (source). 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 can only be downloaded.

## Use Cases

### 1. Share a file

<u>Actor Actions</u>	<u>System Responses</u>
1. Choose “Share” command	2. Displays file dialog box
3. Specify files to share	
4. Specify share groups to share with	5. Show files being shared with a list of share groups shared with
6. Confirm selection	

### 2. Stop sharing a file

<u>Actor Actions</u>	<u>System Responses</u>
1. Select files to remove	
2. Choose “Edit” command	
3. Remove all share groups from the currently shared list	4. Remove file from system.

### 3. Create a share group

<u>Actor Actions</u>	<u>System Responses</u>
1. Choose “Create a Share Group” command	
2. Specify list of users to add to the share group	3. Display users that are about to be added to the share group
4. Confirm selection	

#### 4. Download a file

<u>Actor Actions</u>	<u>System Responses</u>
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

<u>Actor Actions</u>	<u>System Responses</u>
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.

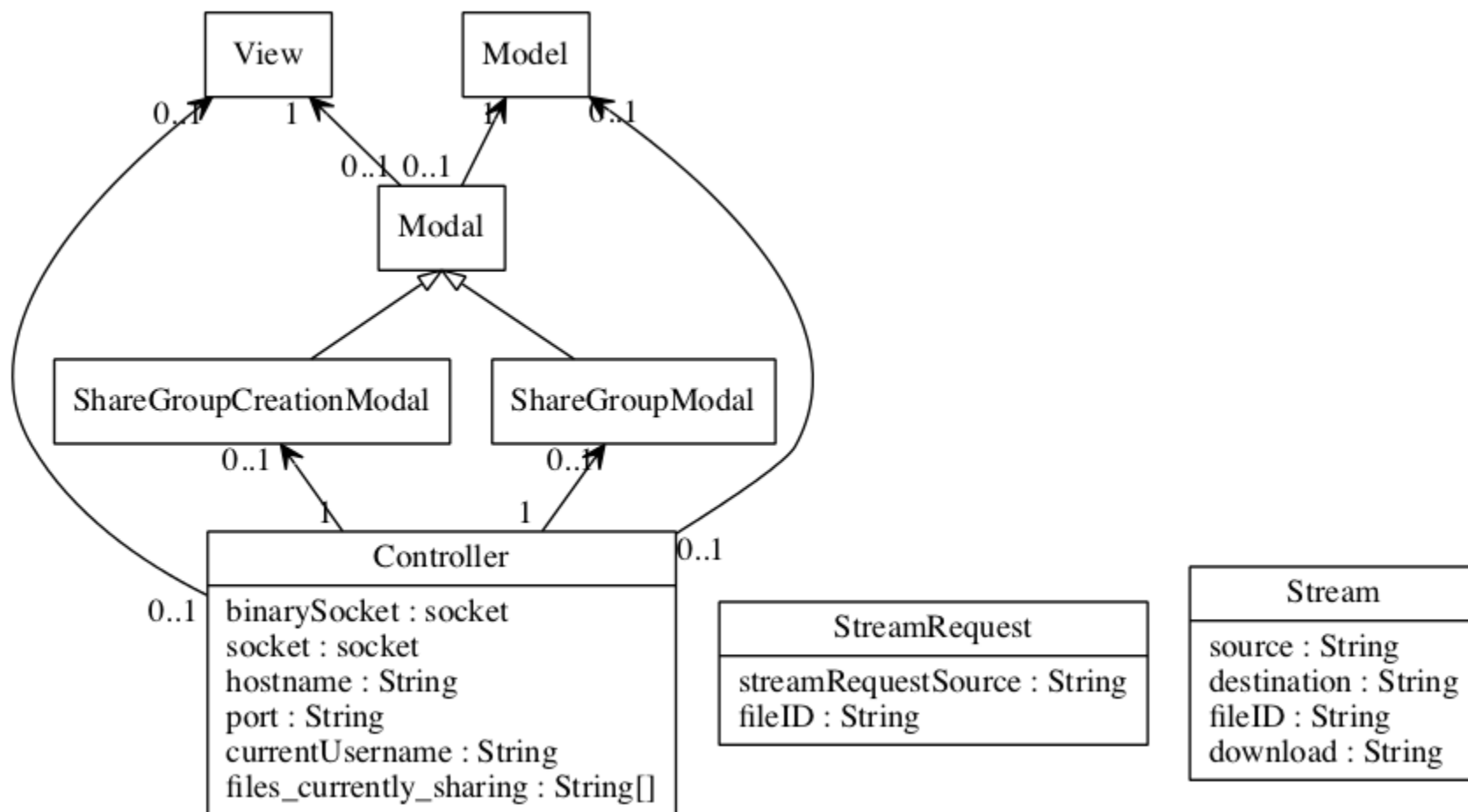
We implement the web application using an HTTP server. We implement this HTTP server using NodeJS and the Express web application framework. NodeJS is in a way, the client-server framework being used for this application. There is a NodeJS server running at all times, and whenever a client connects to 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

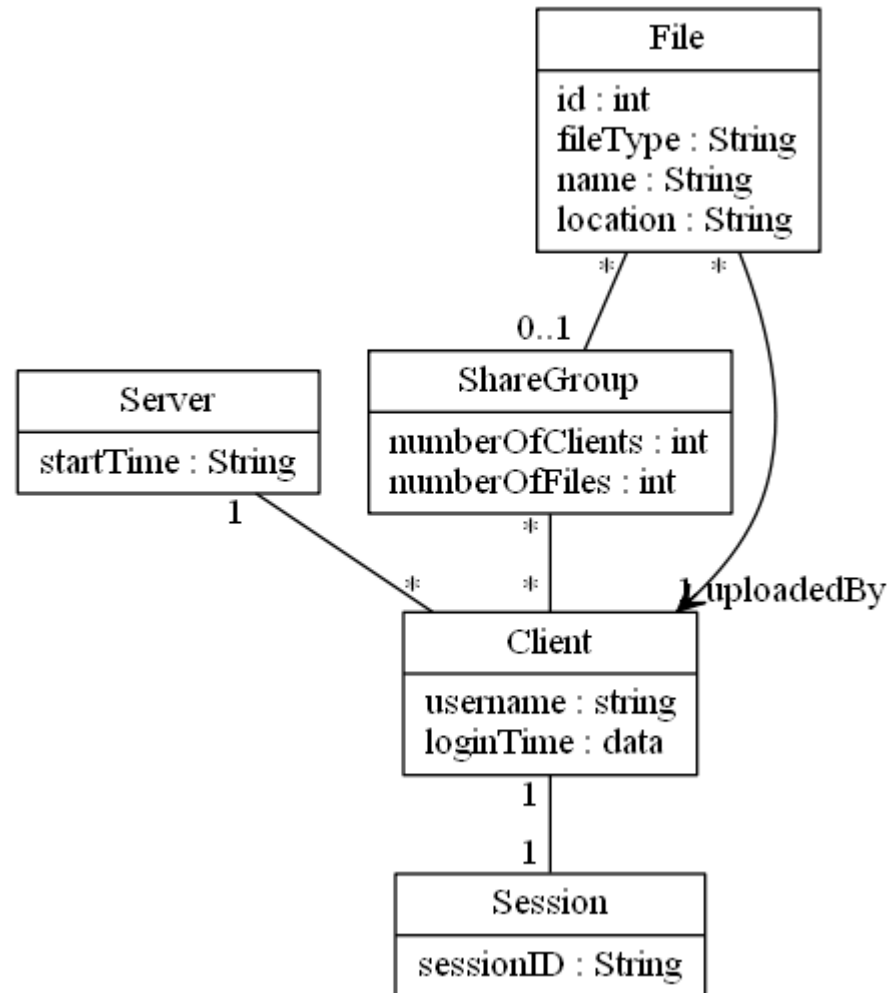
Binary.js is an adapter for NodeJS streams that implements a bi-directional Web Socket that streams uncompressed binary data to and from clients. Binary.js allows the server to provide data or files to clients without actually storing them in a persistent database. Therefore, in this architecture, the 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. Automated test-cases are created using QUnit.

### UML Client Side Class Diagram



### UML Client Side 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>.



### **Additional Instructions**

The automated test-cases are located at: <http://localhost:3000/test.html> (need the server running to make these pass).

The code for the actual test-cases is located in `code\public\js\test.js`. We have a total of 11 tests.

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





































- 1) Extract the attached file “Code.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 Code.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.

Steps for basic functionality:

- 1) Navigate to <http://localhost:3000> with your browser.
- 2) Login with any username.
- 3) Click Share.
- 4) Select 2-3 files.
- 5) Click “Share with everyone” in the modal dialog that pops up.
- 6) Navigate to browse, and test streaming or downloading.

## Git Commit Log

We used Git to track our changes (hosted on GitHub). A total of 60+ commits were made during this project. Here is a very small view of those changes:

	<b>popup windows for audio/video files, fixed sharing and share group ta...</b> ... sarmadhashmi authored 7 days ago	 1f90fdb	
	<b>fix merge conflicts</b> sarmadhashmi authored 7 days ago	 3ab1ab4	
	<b>small view fixes</b> sarmadhashmi authored 7 days ago	 715bb01	
	<b>More bug fixes....</b> Samy Abidib authored 7 days ago	 ab9dc54	
	<b>Various bug fixes...</b> Samy Abidib authored 7 days ago	 760b93a	
	<b>Fixed merge conflicts</b> Samy Abidib authored 8 days ago	 79de6ed	
	<b>TRied some fixes for j issues</b> Samy Abidib authored 8 days ago	 6e7f2ed	
	<b>limit stream button to streamable files and make file ID into a hash ...</b> ... sarmadhashmi authored 8 days ago	 72acfb5	
	<b>added editing file share groups, hooked up download button and show p...</b> ... sarmadhashmi authored 8 days ago	 99b0f06	
	<b>fix client not connected yet error with binary js</b> sarmadhashmi authored 8 days ago	 a0dee8f	
	<b>Non media files are now downloaded and saved</b> Samy Abidib authored 8 days ago	 b86cf6f	
	<b>Fixed streaming</b> Samy Abidib authored 8 days ago	 913ae85	

The full commit log can be found at: <https://github.com/sabidib/FileShare/commits/>