Conclave: Secure, business platform conferencing.

A software engineered approach.

1. Abstract

For this report I have designed and developed a complete conferencing system that will enable organisations to deploy an adaptable, manageable, business communication interface; the name ‘Conclave’ comes from Latin, for “locked room”.

These services will be internally hosted and maintained by the “Conclave” server safely and securely, accessed via “ConclaveClient”, a client side application. Both of these applications will use “ConclaveInterfaces” for all their resources, which defines all the model classes and interaction interfaces. My application will allow for internal one-way video conferencing coupled with a text-based chat for participants. This application will have powerful administrative abilities, with the capacity to manage all connections and conferencing rooms, identify information sent to the server and know where information is going to/from.

Each client side application is generated by the Server, and each contains a unique 32 character AES key that when added to the client application, allows it to communicate with its parent server, and decrypt messages sent in response. This approach will allow for secure traffic between client/server in which prohibited decryption by a middle-man is unrealistic. As each key is managed by the server, it can also be revoked in the case of a client application that is not managed by an approved user, or been compromised.

Throughout the project development, the approach used will be an XP environment, in which testing and development run side by side to make sure that each part of the system is valid and optimized to a few standards which I will outline, before progressing to the next “iteration” of development. As good development is aided by results, several performance logging and mock user frameworks will be created and used to produce testing results and allow problems to be identified and solutions created, as well as services optimized.

Table of Contents

2. Introduction and Requirements

In order to understand the requirements of Conclave, It must be investigated into the state of which conferencing applications are used nowadays, and once investigation is completed: a list of the needs of a conferencing application, and the must-have features that such professional applications should provide. These requirements revolve around an idea of stable, secure service that provides intuitive chat that is easy to use, with powerful administrative functionality.

2.1 Background

Current applications that exist on the market will help in requirements elicitation. A market search was compiled, and the following applications were discovered:

Cisco WebEx

This pay by subscription application was created in 1995 by Cisco in Milpitas, California. This application runs on web applications, embedded mobile applications, PC, Mac, Linux OS applications. This application features functionality such as:

- Full multi-user video, audio and text chat conferencing.
- User validation.
- Administrative controls.
- Record meetings into a specified video format.
- Email invitations.
- Fully integrated browser plugin.
- Whiteboard functionality.
- File sharing.
- 24/7 Live chat customer support.
- VOIP functionality.

Adobe Connect

This is also a pay by subscription application developed by Adobe systems, it was developed in 2012. This runs in Adobe Flash, therefore runs across a wide area of client applications. This application features implementation such as:

- Full multi-user video conferencing, audio conferencing, and text conferencing.
- Powerful plugin API, coupled with a community managed plugin store.
- Screen sharing functionality.
- Connections log and friends list
- User verification service.

8x8 Meetings

This is a pay by subscription application developed by 8x8 inc. 8x8 is a massive media firm that has developed a range of IT technology. It was founded in 1987. 8x8 meetings application features a range of functionality that is integrated across a wide array of platforms, functionality such as:

- Text conferencing and screen sharing functionality.
- VoIP communication.
- Video conferencing.
- Recording functionality.
- User verification service

So after researching these various applications, and their functionality i'll have to decide what I could accomplish in my allocated development schedule time, as well as deciding what makes Conclave unique. I will be focusing on making a business level application, that is intended to be deployed across a business platform. As such, I will be implementing a secure user verification system, that will ensure a user is verified fully and a secure session is built up before any interface permissions are granted. As for interface functionality, the user should always be informed of their state and the servers state by their interface. This will be met in the form of an always updated connections log which will display either all available rooms with their state information, or all connected users in their room; this depends on if the user is in a room. Verified admins should also be able to control all connected users, and perform needed functions on them such as mute/kick/ban, as well as the ability to send them special administrative messages. Asides from user controls, admins will be able to modify rooms on the server, and have the ability to open/close rooms to new connections, as well as create new custom rooms of a specific type and with a password.

As Conclave is to be deployed in the workplace, we must try and understand any specific requirements this domain would have on a service like Conclave. Managers would need to inform users of announcements or set meeting times that all connected users will be able to see, this would help the coordination of their workforce. Users should be also able to save chat logs into a more permanent form, as this would help participants recall things that were brought up and discussed in the chat. In terms of the Video Conferencing requirements, I would expect that the functionality to take screenshots of the current streamers stream and save it to their local file would help users also recall valuable meeting information. After assessing all intended uses of Conclave, I was able to compile a list of functional, and nonfunctional requirements that will be implemented throughout development:

2.2 Functional Requirements

- Many types of ConclaveRooms can be hosted on a server and any users connected must be constantly informed of room status changes: such as number of users in a room vs maximum capacity, open status, room type and room name.
- Clients should connect and start to send login information securely and be protected from possible man in the middle security (MITM) attacks such as packet sniffers or spoofing attacks.
- For this implementation of Conclave, there must be two types of rooms implemented: a standard TextRoom, with just ChatLog functionality, and a ConferenceRoom which extends TextRoom and implements the ability for one user at a time to stream their webcam to all connected and viewing users in that room, as well as the ChatLog.
- When joining a server, all users will be presented with a “Frontpage” which is always visible to users that are not currently in a room, and will display all server announcements when they appear. Admins will post announcements.
- Users can join any open room, and if that room has a password; be presented with some kind of input service.
- Users can send private messages to any other user that is in their room, these are only seen by the recipient and the sender.
- Once inside of any room, users that are not muted can post in the room chatlog, and all users inside the room will receive that message in real time.
- Admins will be given the correct interface upon login, which will allow them to perform operations on rooms such as: Add/Remove rooms from the server, Open/Close rooms to new connections, mute users from chatting and streaming their webcam, kick users from rooms and also ban them from the server.
- The system will handle disconnecting clients gracefully, and ensure that all clients the server shows are actual connected, and interacting clients.

2.3 Quality Requirements

- Private Rooms and Users are persisted into a Database, with their passwords stored securely in a hash code, combined with an added protection of a random salt.
- Users should be able to export their ChatLog at anytime, which will save it to their local ConclaveClient directory in the form of a readable .txt file.
- Users connected to a ConferenceRoom can take screenshots of the current active stream at anytime, this will save it also to their local ConclaveClient directory in the form of a .png.
- Users should be able to control their Chat Log message display entries and choose to filter out certain message types such as Admin, Private, Room, and System.
- Users should be able to refresh their connections with an input, it should update automatically but this will allow Users to verify their own state whenever.
- All administrative interactions with the server are to be logged using a Logger implementation, which will log useful information such as usernames and parameters. Logger will also log users logging in and out and any exceptions the server throws.
- All server overheads will be logged into a suitable .csv format, writing such details like CPU performance, memory overheads and thread count, along with server states like active room, logged users, and anonymous connections (These are connections that have not logged in yet).
- Admins should be able to send one-way messages to any connected user of the server, this is intended to provide a warning or to inform users of a room password.

2.4 Frameworks and API's

For room and interface functionality, conclave will use Java's Remote Method Invocation, with registry policies, to broadcast rooms and User/Admin interfaces. This will allow bi-directional communication that is ultimately managed and controlled by a central server. This approach is also distinctly Object Orientated, and conforms with the Java design principles. For account management login services, I'll be using Java.net ServerSockets. This will allow users to connect to a specific socket and pass requests in the form of UTF-8 text. This implementation will pass back a HTTP response code on the state of this transaction, which will can pave the way for potential future 'Account' orientated interactions using servlets and browsers. To implement its video conferencing ability, we will be using 3 different frameworks working in synchronism:

Sarxos Webcam-Capture. This framework allows the Client application to locate an appropriate webcam and capture BufferedImages. Through the use of Timed Event we can capture an image every frame and pass it to the encoding/decoding framework.
Xuggler. This framework allows for the system to encode BufferedImage into h264 format frames, it also serves as a decoder for h264 into BufferedImages.
Netty IO. This powerful framework allows for channels across a network to be created which will hold frames from its streamer and place it in its buffer to pass it to listening clients in order.

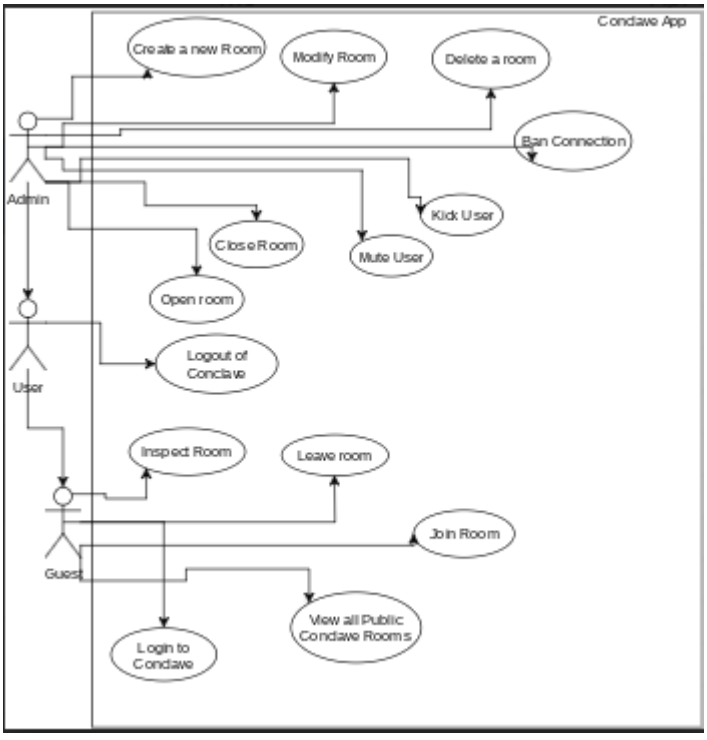
3. Design and Analysis

3.1 UML Use Cases

In order for me to assess all the potential actors and interactions of the Conclave system, A Use-Case diagram was built that adequately captures these. The next figure is the first version of my Usecases, and I had originally anticipated that conclave would need a “guest” like interaction, but after making a development shift away from commercial features like ‘guest’ which is insecure and provides less administrative control; and towards secure communication and wide control of connection environments. Notice also that originally there was no motion of client keys or room password validation; this is also due to the paradigm shift of my development and the result of security orientated milestones.

V1

[Fig3.1]



V2

[Fig3.2]

Note: Missing from this diagram is the ability for admins to close/open rooms; this was added in a testing iteration but does count as a use case.

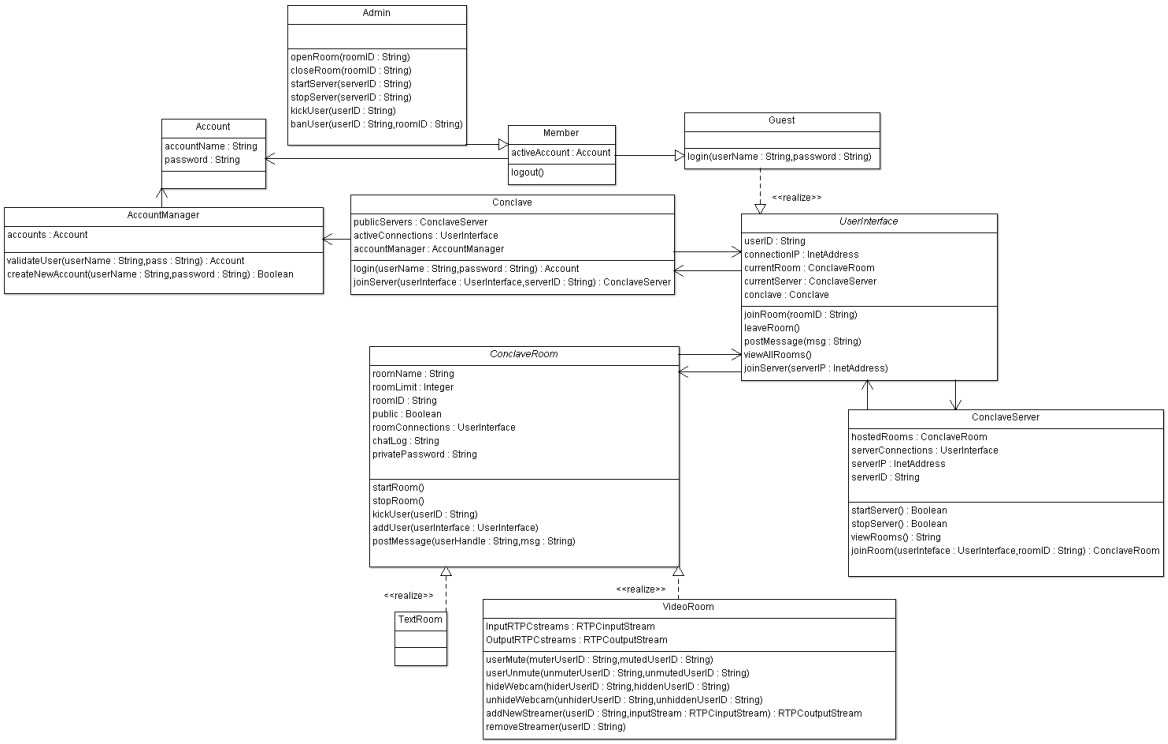
3.2 UML Class Diagrams

For an ultimate look into the system, it became necessary to produce a complete Class diagram of the server, client and any shared resource project that contains all the needed interface classes and relevant model classes, these are to show the various objects and the encapsulation of each object within the system. At the start, a rudimentary server class diagram was constructed to aid development, this proved useful in order to establish the basic design of the system and ease development. After development had completed, a second class diagram was constructed; this shows the accurate view of all the classes in the final release of the system; and will offer a glimpse into the implementation of the system. There are 3 different types of Class diagrams constructed: Server, Client, and Interface.

3.2.1 Server

V1

[Fig 3.3]



V1 of the class diagram did not develop a Client Application, and is an indication of how early in the development this was designed. The final class diagram ended up being incredibly more complex, and many new classes that were not foreseen had to be developed

V2

This next class diagram was designed with a few patterns in mind, such as the Factory pattern for objects that are initialized differently like the Room and Account, and SessionKeys. These active objects were also managed by these factory classes. We also have used a Façade pattern in the form of a ServerController, which is responsible for interacting with the server state and all its elements to it such as Registry loading, password hashing and client handling:

[Fig 3.4]


RoomManager
AdminInterfaceImpl
SecurityHandler
UserInterfaceImpl
AccountManager
ServerController
Account
Sessionkeys
Userkeys
PerformanceLogger
ServerTerminal
ClientHandler
main
ConferenceRoom
TextRoom
RegistryLoader
RMISecurityPolicyLoader
Encryptor

3.2.2 Client

A Client diagram was also created, to show objects involved in displaying the conclave state and room interactions, as well as streaming and listening to webcam channels. Separate interactions are kept in different panel classes, and are slotted into the main GUI when they are needed or authorized. This allows a flexible design that allows future panels to easily be implemented, as they can slot into the TabbedPanel frame container, with an identifying tab name:

[Fig 3.5]

3.2.3 Interface

This interface project also contained model classes to be used in the Server and Client programs, they follow a hierarchical pattern in terms of inheritance; as each interface builds onto another interface providing a different layer of encapsulation. ConnectionsLog and ChatLog provide model classes for output events like messages or system notifications, room opening or players connecting/leaving a room. By having this in the interface project we can use these classes to initiate a state from a point in time:

[Fig 3.6]

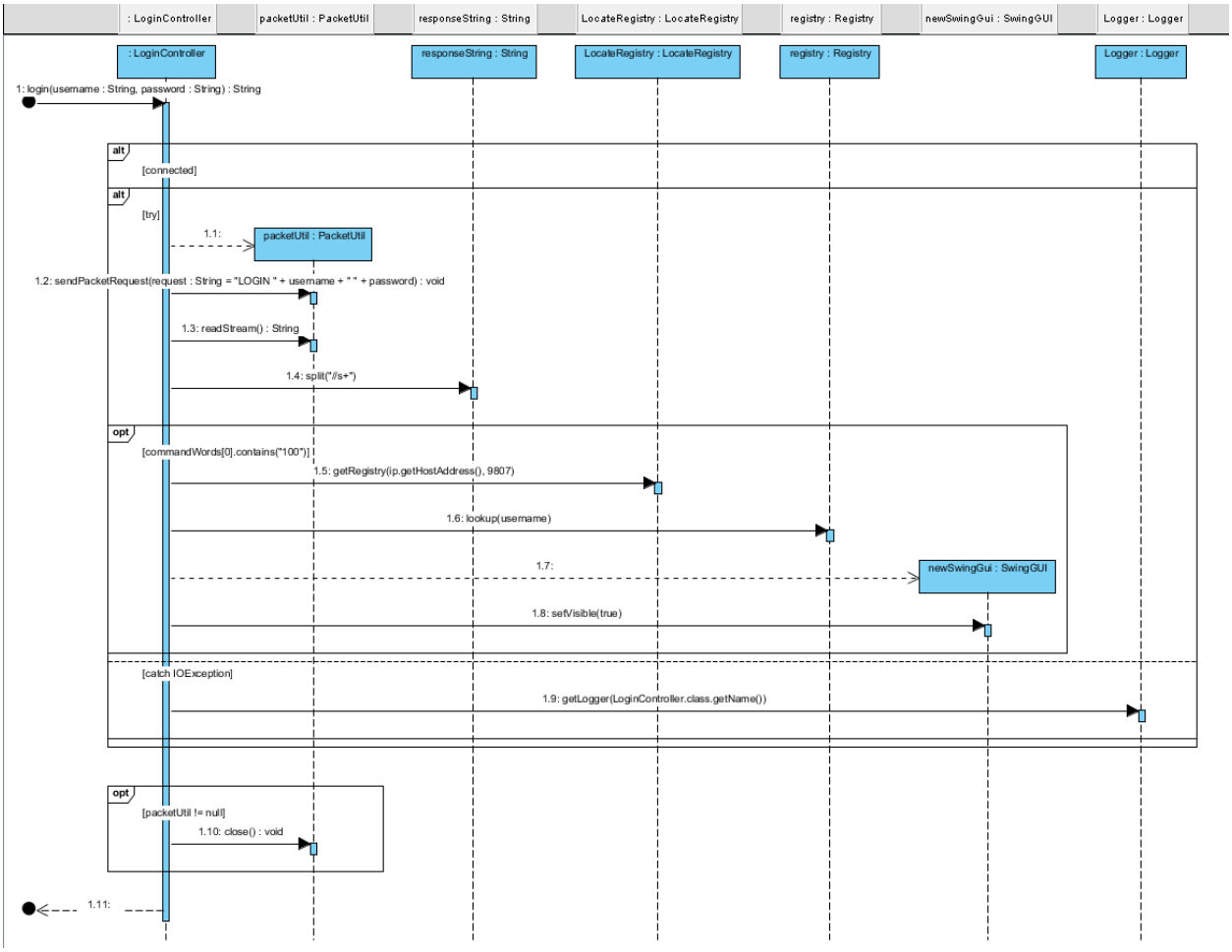
3.3 UML Sequence Diagrams

In order to understand the a few complex interactions with the server, a low-level like sequence diagram was created for a few conclave interactions, these are intended to show how the system interacts with its various modules.

3.3.1 Login Sequence.

This next diagram illustrates the sequence on the Client side of the system when a Login request is made. This diagram only encapsulates the client side interaction of the deal, as this exchange has an asynchronous like processing, with the client side capturing user input and reading a request; then finally locating the registry and retrieving a RMI hosted userinterface implementation object and beginning the Swing session.

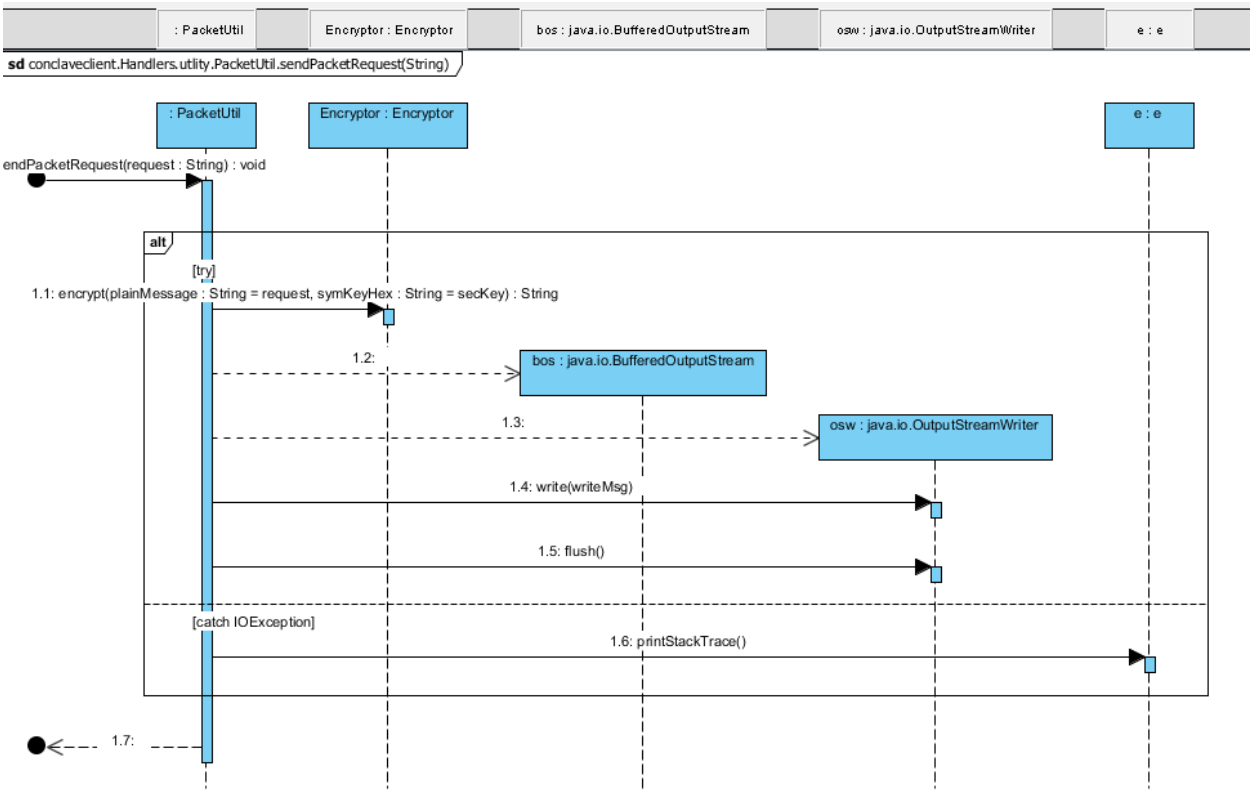
[Fig3.7]



This sequence diagram does not show the Encryption level, in which the client uses an encryption static method call to AES encrypt the message to be sent, this method takes a SecretKey string, which is initiated before login can be started, and a String to be encrypted. It in turn produces an enciphered text

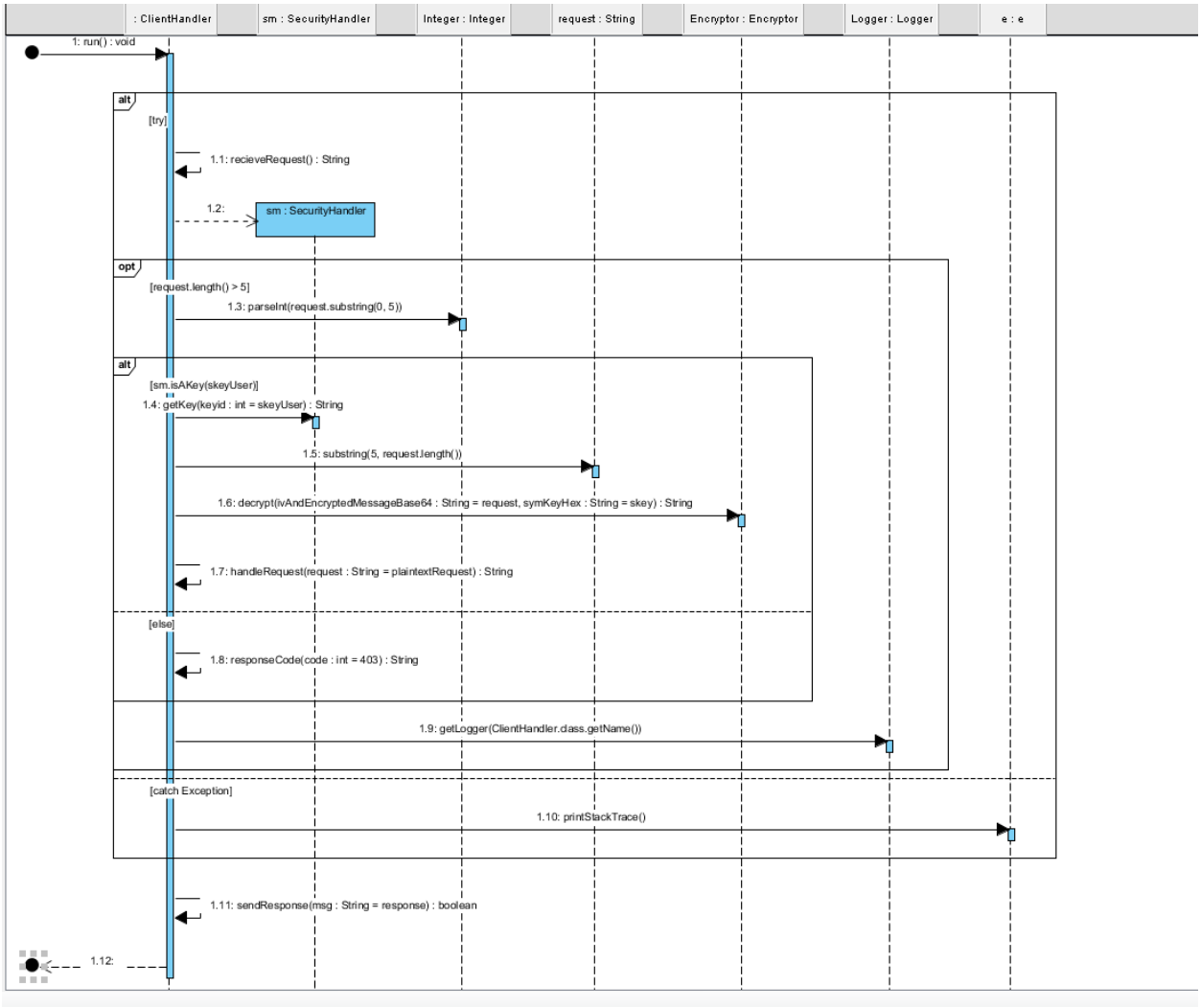
that can be decrypted using the very same key. It then concatenates the enciphered text with a KeyID integer, in which the database looks up and uses that key for its decryption. The next sequence diagram will show this interaction:

[Fig 3.8]



After this request has been sent, the next diagram will illustrate how the server deals with this request, including decrypting the message and handling its response by looking up conditions and verifying credentials before persisting to a database or mounting an Interface object to the Registry. This ClientHandler will also return a response code, for the client systems use, as well as a general purpose status output string that the client can display or log:

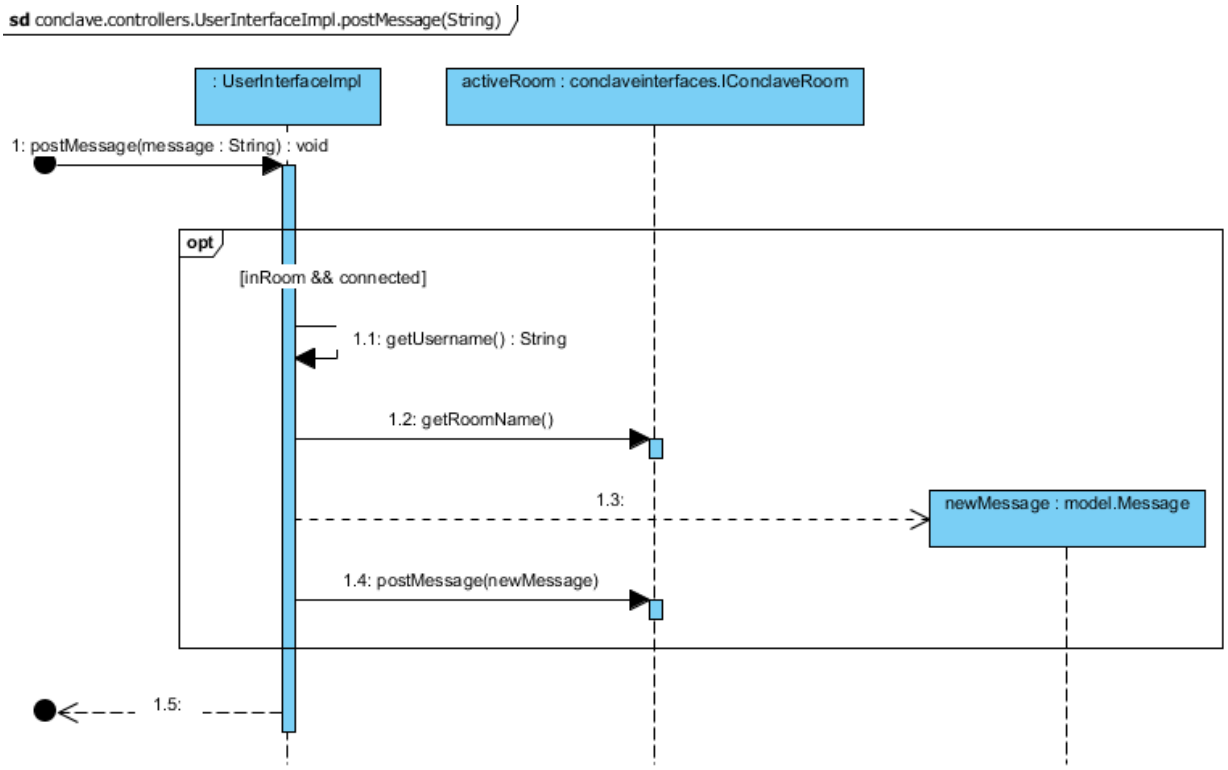
[Fig 3.9]



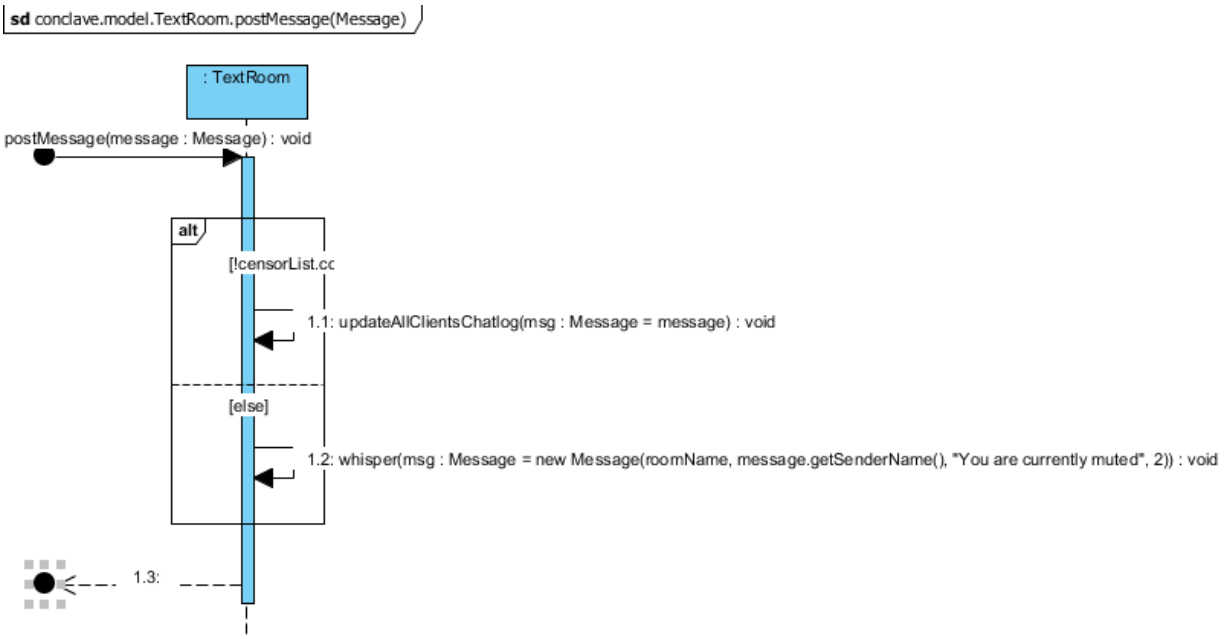
3.3.3 Conclave Room, posting a message.

In the next set of sequence diagrams, we will be outlining the sequence involved in a user posting a chat update in a room, from the UserInterface object posting a string message to each participating client receiving an update. In this example we will use a TextRoom object as the room the example user is current active in:

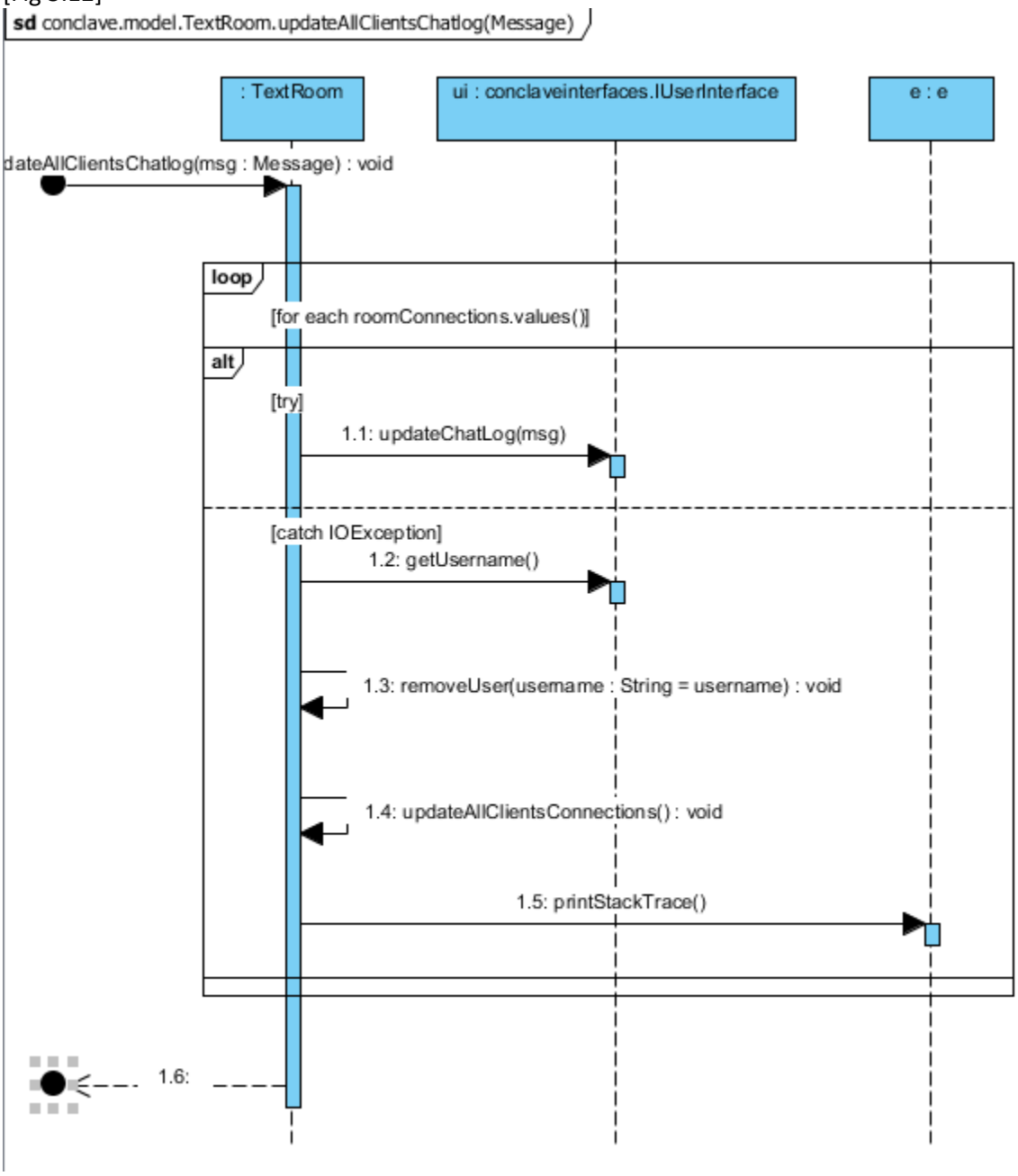
[Fig 3.10]



Once the TextRoom has received the message, it will then update all clients chatlogs, or inform the user that they are muted and cannot post in this room: [Fig 3.11]



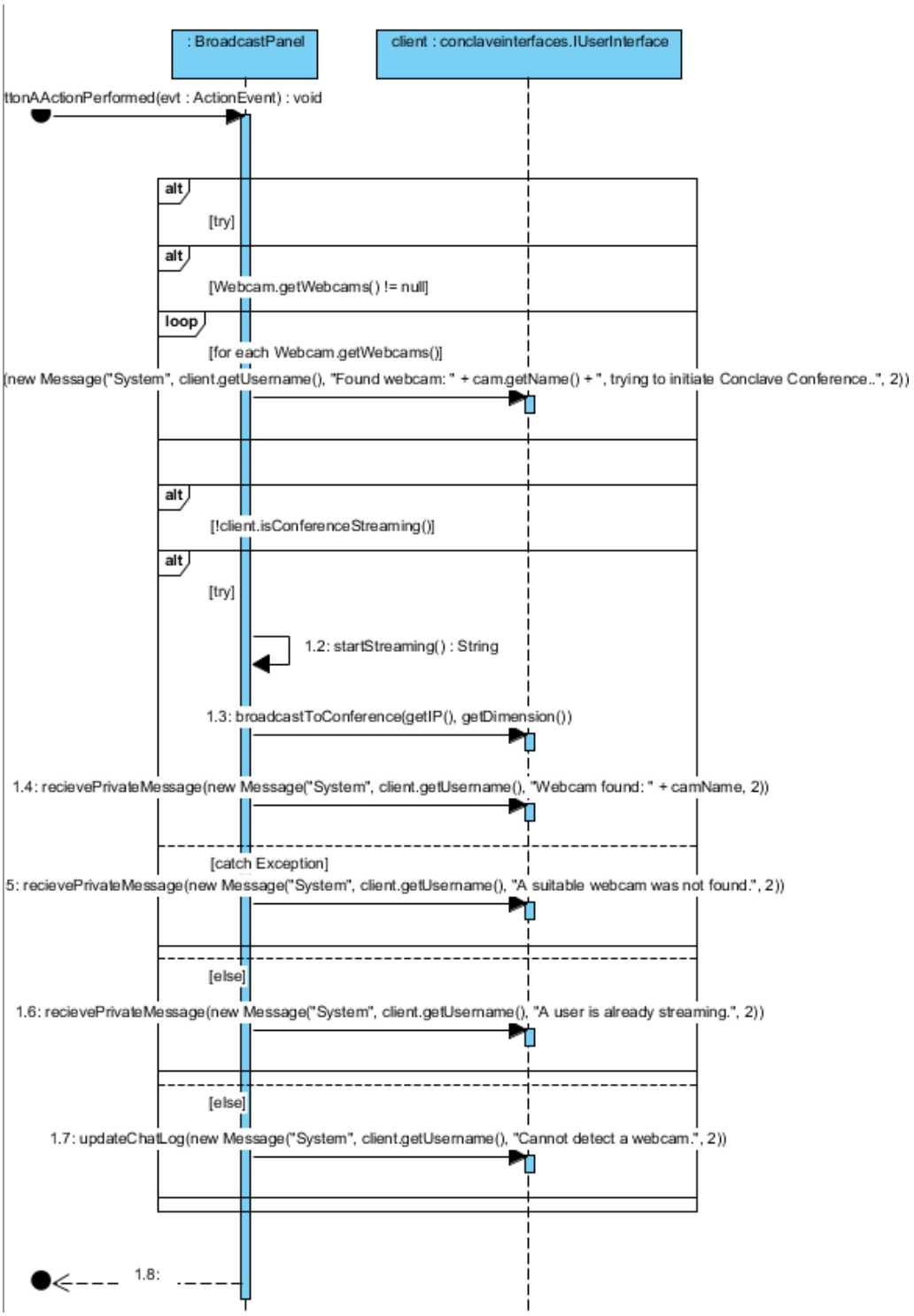
Updating client's chatlog also serves a dual purpose of helping keep the connections array clear, and by removing clients that throw a remote exception we can try to reduce inactive players appearing in connectionslog or lingering in the system:
[Fig 3.12]



3.3.4 Conference Room Streaming.

In this series we will be investigating the interactions that occur when a user begins streaming their webcam, and how each object interacts with one another to achieve asynchronous frame updates. The next sequence diagram describes the client locating a suitable webcam, and if they are allowed to stream, before broadcasting to the conference, initiating a StreamingServerAgent:

[Fig 3.13]

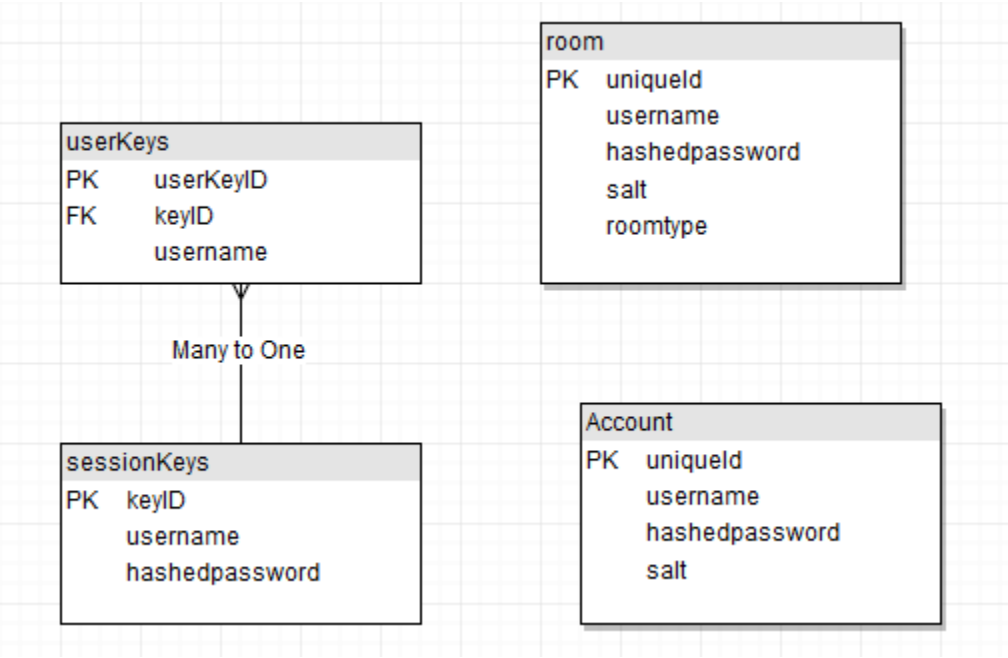


Once the StreamingServerAgent adds their webcam to the channel, a TimedEvent will take a BufferedImage over each frame, and pass it into the netty channel buffer upon which all connected clients will redisplay their video bean with the new image, after it has been decoded.

3.4 Database Design

As the server is to use a database to persist its objects, it is imperative in the name of good Software Engineering to create a database schema diagram to show each table and its entries. So, an EntityRelationship diagram was constructed to identify entries, this diagram also denotes primary keys of each dataset, and show join table relationships like many to one. This diagram will also help aid development by helping construct an SQL statement to initialize:

[Fig 3.14]



3.5 UX Design

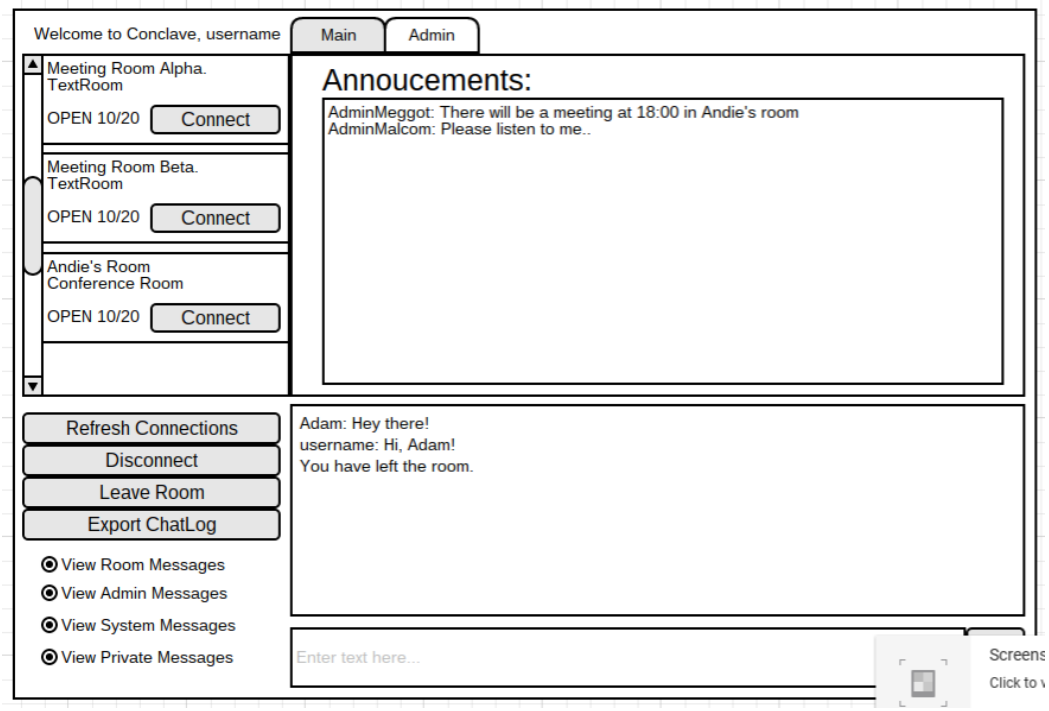
In order to aid development of the frontend part of Conclave, we should design a User Experience diagram that will illustrate the basic wireframe of the GUI to follow. This GUI must conform to a few requirements on the system, such the user being constantly aware of their state and any interaction should provide instant feedback. This was met in the form of a GUI that uses the same area for interactions such as Connections, or available rooms. The Chatlog provides an area where users can talk to each other, and this area also provides intuitive feedback from the system by defining each message as different in the form of Message types, such as System, Room or Private messages. The View can decided to display these messages differently to enhance visual feedback.

As for customisation of Conclave Rooms, by employing a TabbedPanel approach to the main display area, we offer potential real-estate for any new room interactions. This is utilised in the ConferenceRoom, which takes advantage of this panel to show a Broadcast Panel, which in turn defines

specific Broadcasting interaction buttons. The next wireframes show what the user should see when the system has completed. These wireframes were created in Moqups.com:

3.5.1 Server Frontpage

[Fig 3.15]

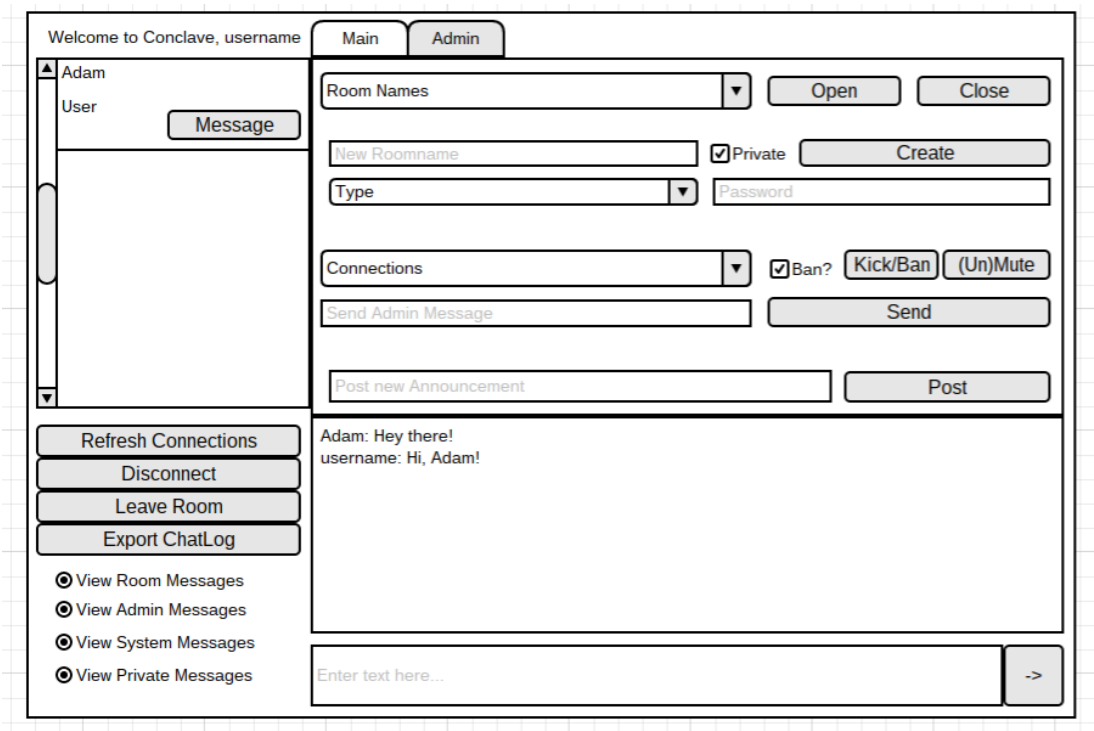


3.5.2 Conference Room

[Fig 3.16]

3.5.3 Administrative Window

[Fig 3.17]



3.6 Designing Tests

As we are employing an XP approach to this project, we must outline testing criteria for the development to use throughout. A service few milestones have embodied this and been set for the system to achieve in terms of reliability and efficiency:

- *The system must deal and handle all possible inputs from the client with appropriate responses.*

To address the first criteria, we will take a whitebox approach to testing. The development will use JUnit style testing on the end interface, logging expected results against actual results. This will employ Use Cases in its testing to evaluate development progress using requirements and end user cases. By using carefully chosen input cases we can hope to identify outright bugs early-on.

- *A system must reduce its overheads whenever it can, and allow itself to perform effectively under high load.*

In order to reach this criteria, it was decided to adopt a performance logging approach that would hope to aid development by producing quantifiable evidence of why certain design decisions were made any and provide results that any optimisation decision produced desirable results.

- *The system will keep up to date logs of exceptions and any chosen server interactions*

Not only will this milestone affect design of the system, but this als o will help in designing tests by identifying exceptions and allow tests to be constructed that mimic these exception cases, and allow a solution to be developed. Exceptions will be logged during development, allowing problems to be identified in the report as well as any novel solutions.

3.6.1 Integration Tests

Integration tests were built using JUnit, each test was scoped to test exception cases and performance issues of interface interactions. JUnit tests results were added to a spreadsheet with each test numbers labelled to show test flow. In order to repeat the integration tests, you must also first initialize a few variables such as room names, server information, account information and such; to stimulate a consistent result. This was achieved through the usages of JUnit @Setup and @Teardown annotation methods which allowed me to set up the space before each test.

3.6.2 Performance Tests

In order for conclave's performance to be logged and analysed, it must be found a way of reliably logging all the overheads and performance data that will be of relevance for later on improvements. In order to do this, I have decided to use a csv like logging mechanism that will save useful variables like allocated memory used, thread count, average network response time, connections during tick, and logged-in users. This performance logger will run in a concurrent thread that will save an entry every defined server tick. This will allow me to refer to specific and real results during development and visualize any changes in system performance.

3.6.3 Server Load Tests

In order to understand how Conclave performs under moments of high stress, a framework was created that will simulate many users connecting to the server at one time; logging their response time and then finding out the average response time from all number of iterations. This program will allow the user to test an array of different request types in one test. This load testing program is intended to be used on a separate computer within the same network, to simulate how Conclave is to be used in deployment.

4. First Iteration Development

The first iteration will implement a primitive text chat room, a login/create account service as well as provide the basis for functionality that will be developed later on in the development lifecycle.

4.1 Interface Defining

Before implementing the system, we must define the interface classes that the system will be implementing throughout. By doing this before writing any frontend or backend code we can separate their development, as backend should should provide the interface implementation, and the design to interact with it. As Conclave also uses RMI's for its Rooms and Users, we must make sure that we follow Java's rmi convention by making sure that firstly all interfaces throw a RemoteException exception, and that eventually all objects the implementation uses can be serialized. In terms of the methods these interfaces should define, we must always be thinking about bi-directional communication: that the server will also use the interface to communicate with users, and vice-versa. We must also remember that the interface will not own the users view, and should merely model interactions and provide stateful information to objects that own it.

Defining the User Interface.

The user's interface revolves around five services that every user must achieve: viewing connections statuses and interacting with them, joining/leaving available Conclave Rooms, posting and receiving messages either to other users or to an active room, the ability to receive server frontpage updates, and finally broadcasting their webcam to a ready conference room. In order for the end client display to display information, it was decided to use boolean return methods to see if a part of a server or room's state has changed, and if so; the display would then request the appropriate information. By using this compartmentalized design, we can reduce unnecessary RMI TCP traffic by only requesting the information that the client does not already have.

Defining the AdminInterface.

After the User Interface was defined, we can build upon this by defining an AdminInterface which extends this design, providing control methods on the server that the requirements has defined. By extending the User Interface design, we can forget about potential runtime casting errors, as each User

Interface that is added to the registry will always implement at least UserInterface. By calling a getType() method on the interface, we can find out what implementation we are using without using Java's reflection library, although this would also be a valid approach.

Defining the Conclave Room interface

What functionality each Conclave Room is to use. Each room should provide basic user interactions, such as the ability to join/leave each one - adding their interface to a room collection for usage in maintaining a Connection Log and distributing room messages to valid users. Asides from these update methods, we must define control methods on each room, that the AdminInterface will use to control active users and its room state. Extending on this interface is also the Conference Room interface, which defines methods for streamers/viewers to use to receive or post broadcast information, such broadcast information like streaming dimension, IP and Port.

4.2 Object Managers. Persistence and Validation

Conclave's management and creation of persistence held objects is achieved through a Factory pattern. Such classes like AccountManager and RoomManager fulfill this pattern by creation of property filled active objects and updating their state if needed, as well as persisting them to the database. As these managers must update and control one set of objects, we must also employ the singleton pattern here to make sure that each instance of managers is kept the same. The databse schema was developed according to the Entity Relationship diagram, in which the 4 proposed tables were to be created in order to store active conclave elements like rooms, users, secret keys, and userkeys.

SQL was used to generate the database, which in turn generates the Entity classes encapsulating the database objects. This SQL also selected primary keys for each table, and in one case should select a foreign key, linking it to another database in a many-to-one relationship. By using AUTO-INCREMENT commands we can create a unique primary key for each table, this key starts at 0 and increments by 1 every time an object of that class is persisted, each entry also provides a maximum length for each entry. The SQL is provided below:

[Fig 4.1]

```
CREATE SCHEMA conclavedb;

SET SCHEMA MEGGOT;

CREATE TABLE account (
        userID INT NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1),
        username VARCHAR(15) NOT NULL,
        hashedPassword VARCHAR(80) NOT NULL,
           salt BLOB(60) NOT NULL,
        CONSTRAINT userID_pk PRIMARY KEY (userID),
           UNIQUE(username)
        );

CREATE TABLE room (
        roomID INT NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1),
        roomName VARCHAR(15) NOT NULL,
        hashedPassword VARCHAR(80),
           salt BLOB(60),
        roomtype VARCHAR(3) NOT NULL,
        CONSTRAINT roomID_pk PRIMARY KEY (roomID),
           UNIQUE(roomName)
        );

CREATE TABLE sessionKeys (
        keyIdentify INT NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1),
        keyString VARCHAR(45) NOT NULL,
        CONSTRAINT keyIdentify PRIMARY KEY (keyIdentify)
        );
CREATE TABLE userKeys (
        userKeyID INT NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1),
        keyID int NOT NULL,
        username VARCHAR(15) NOT NULL,
        CONSTRAINT userKeyID PRIMARY KEY (userKeyID),
        CONSTRAINT keyID FOREIGN KEY (keyID)
        REFERENCES sessionKeys(keyIdentify)
        );
```

Creating and Persistence Accounts at Runtime.

Accounts are created by an anonymous user through the ServerController by their ConnectionHandler thread (Which receives HTTP like requests). As the connection thread uses the ServerController to create and login accounts, it allows the RMI registry of such objects to be separated from the persistence manager. This allows us to also deny connections of certain users if they are on a banned list, and also log connection attempts into a centralized logger. Once the account is created, it is persisted to the database.

Creating Rooms at Runtime.

Rooms are managed in much the same way, however we have the added requirement of these managers to build a connection status display and update to show correct room statuses whenever certain things happen: such as incrementing the active users or opening/closing rooms. These connection statuses have been modelled in the form of a Connection Log, which contains entities of a

Connection Entry, providing information such as description and a name. When a user is connected to the system, they are given an instance of this Room Manager, and can use it to receive an updated Connection Log whenever they leave a room or when a room state changes. If the room contains a password then it is persisted into the database, rooms without passwords, or so-called open rooms ,are not persisted as this was deemed a wasteful investment and is counter-intuitive to the Conclave system. Persistence is handled by a RoomManager, using the EntityManager.

Encryption Methods

Conclave stores passwords in the form of SHA-256 hashes, which are hashed along with a random salt created by a java.security.SecureRandom object using SHA1PRNG instance. This hashing adds a layer of security by mitigating the potential security damage of having all a system's passwords stolen in plaintext. By adding a unique salt to each password, we can further protect user passwords against most rainbow tables, which enable hackers to crack a bad password hash in a matter of moments.

4.3 Logging in and maintaining valid connections

Using Java's server sockets we can manage accounts and anonymous connections that are eventually switched over to java's RMI functionality when an appropriate controller object is created. This distinct approach was chosen in order to separate the two different parts of the system, and thus Accounts could be used for other program usage as they are serialized into a non coupled database.

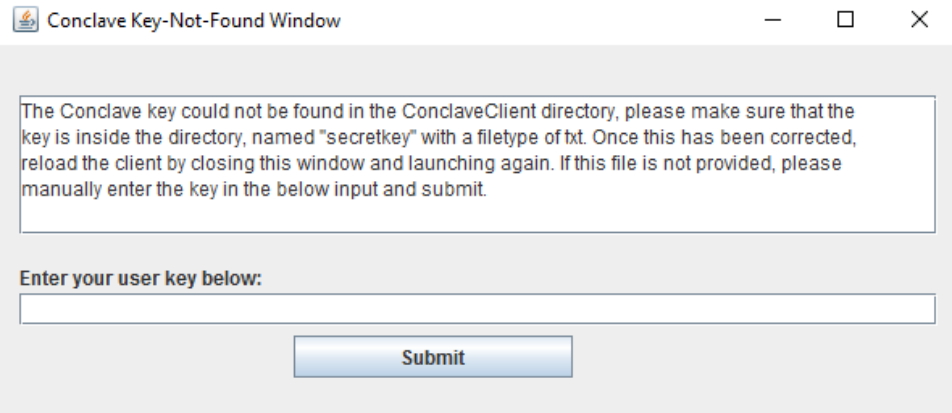
Secure AES connection platform

In order to send such sensitive information as usernames and passwords over a network, we must ensure that the traffic is encrypted and decrypted securely. For my implementation, it was decided to use a pair of AES keys that the server generates for each new client's JAR, and persists it to the database with a keyUser identifier. The generation file contains a secret key which it will use to encrypt and send its account information, the server will then decrypt the message using the appropriate key pair and handle the request, before sending back the response; which is also encrypted. The server can then manage these keys and render certain ones invalid, if they have been compromised for instance.

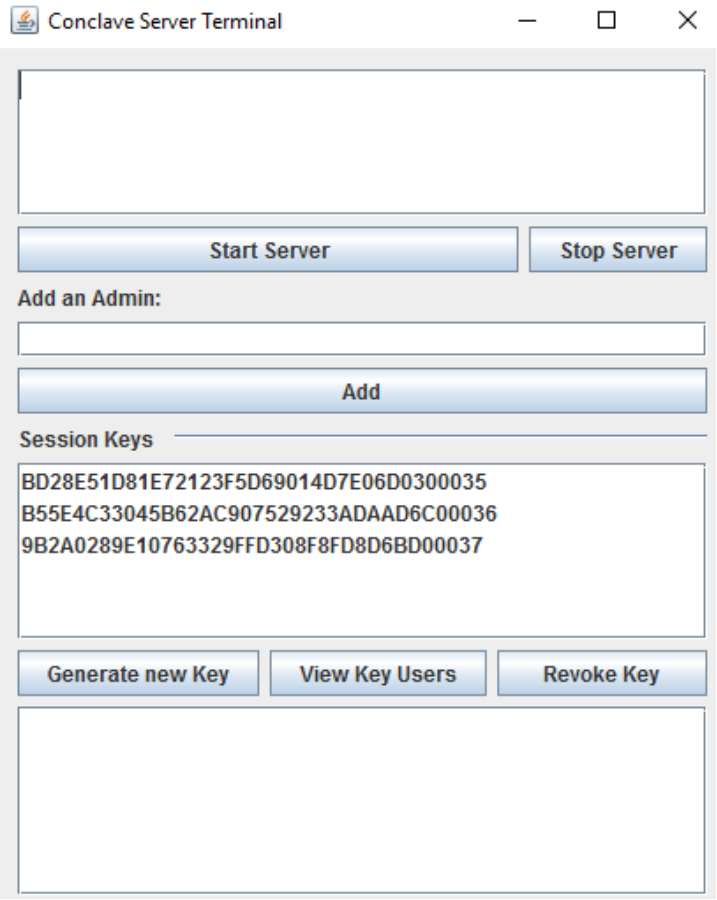
It achieves this through the usage of the SecurityHandler class, which accesses the database to retrieve "SessionKeys" when a user sends a request through. The client will make sure to put their identifying integer, or userKeyID, appended to the start of the message, this will provide the server with an index to lookup in the database to retrieve the appropriate key to encrypt/decrypt traffic. When a user logs in, the server will check to see if the username has ever been logged using that key, and if not then it will create an entry in its join table "Userkeys" which will input the secret key id against the user ID.

The client will load their SecretKey by the client program loading in "secretkey.txt" in their directory, if the file cannot be found a window will appear asking the user to either place the file in their directory or input their secret key manually through an input box. This window will also provide appropriate outputs as to whether the key is in the correct format. It will not check to see if the key will work on the server,

but it will save the inputted text as the txt to prevent further hassle. If the key a user uses is invalid, the server will respond with an appropriate error message telling the user the key is invalid. The final implementation of this window can be found below:
[Fig 4.2]



Through the server terminal, a super admin can choose to generate new keys for distribution, they may also view active session keys and view a list of all the users who have used that key. They also can decide to revoke keys which will delete it from the database and prevent any client using that key to connect to Conclave. The server terminal was modelled as a JFrame which loads when the main method is run, allowing the user to start or stop the server, and add admins:
[Fig 4.3]



Logging In

As explained in the frameworks section, logging in is controlled by a ServerSocket and connection handler threads for each connection that has not yet been verified and logged in. This serversocket is run by a ServerManager runnable object, which handles the serversocket.accept() in its thread; which in turn creates a ConnectionHandler thread to prevent the server from freezing up. The ServerManager will also log this request as a request per tick, which the ServerController will use in performance analytics and management. Next is a table which will show the valid requests a server can receive, as well as their parameters and finally, a response sent by the server; depending on if the transaction was successful:

[Fig 4.4]

Request Name	Request Parameters	Response
PING		Server is alive!, Invalid key given
LOGIN	Username, Password	Valid Login, Invalid details, User does not exist, User is already logged in, Username has

		been banned, Invalid key given
SETUP-ACCOUNT	Username, Password	Account Creation successful, Username already taken, Invalid password/username, Invalid key given

In the future, more requests could be added such as the ability to return information about the server using GET (resource) request. This could be used to return information like how many rooms there are on the server, what rooms the server supports or how many logged in users there are.

Creating Connections

Once a user has logged in the ConnectionHandler thread is terminated and the socket is closed, calling the ServerController to register an appropriate User Interface to the RMI registry with their username as the key. Once the User Interface is registered, Conclave must maintain this connection in its collection and if the user disconnects or logs out, it must remove it from this collection to maintain a valid list, as well as to unregister the object from the registry. Once the client has connected with their GUI, they are then marked to receive updates to room status and also frontpage announcements and private or admin messages.

Managing Connections

It achieves this through a TimedEvent object that runs when the server starts up, which every so often (1 Server tick/1 Second) will iterate through the collection and call a isConnected() method which if it returns true then the user has logged out, if the isConnected() throws a RemoteException, then the user has disconnected; in any case, it will then remove them from the collection and also from the registry. This thread will access the server's collection, and we may encounter certain concurrency issues.

To avoid ConcurrentModificationExceptions, which happen when two separate threads try to access and edit a collection at the same time, there are many solutions. On the ConnectedUsers array, which holds all connections, I've decided to use concurrency safe collections: such as ConcurrentHashMap and a CopyOnWriteArraylist which copy collections before writing and then saving when the collection is available. I could also use latch and notify() blocks but in this case, is not necessary.

4.4 User and TextRoom implementation.

To reduce potential risks of a malicious user misusing Conclave, I decided against the conventional RMI practice of the standard user implementation to contain an instance of the Server. This is because, for basic user and admin interactions, the user only needs to be used by the server and not the other way around.

User Interface implementation.

The user will then contain an instance of RoomManager to supply rooms interaction, and an instance of the room it is currently in to control active room interaction. To join a room the interface will lookup the registry by roomname which acts as the registry key. It achieves FrontPage updates through its own held FrontPage which the ServerController updates it. The object also stores the ChatLog, which contains a collection of all Messages it has been sent. The client will frequently download them from the user and display them in the view.

TextRoom implementation.

The Conclave Room implementation, TextRoom, is responsible for updating all connected users state whenever there is an update. It does this by iterating through it's connected users hashmap and updating their chatlog/connectionlog. This will also remove the user from the room if a RemoteException is thrown, keeping its ConnectionLog valid. Messages are passed through these rooms and to the user using a model class, Message, which defines message information such as recipient, sender, type and contents. This enables the user to be able to filter out certain message types and also for the room to decide if a message is a private message or a room message. This implementation class is also responsible for overriding the getInfo() method, which the ConnectionLog will use to display room information.

4.5 First Iteration Testing

During the first iteration we have implemented many features, with these features we have developed a rudimentary account management service that enables users to login and create accounts. We have also developed a rudimentary chatroom service that allows users to join/leave rooms, post messages and receive connection information. Using these classes, we can begin the first cycle of unit tests that will help us identify early issues and improve development by providing a tested and valid Conclave framework to build the next iterations upon.

Login Service Unit tests

After this server state was initialized with the testing variables, unit tests were done to the system. These tests should be reproduced in the order they appear as some tests results require some state changes to have occurred, such as testing to see if Login will allow the same user to login multiple times without logging out. In order to reproduce these tests, the following variables must be imitated as show:

Testing Parameters
Server Online at 192.128.0.3:20003
AdminAdam setup as an Admin
Conclave Version: V3.0

These tests were performed by a Junit class, and of which's source code you may find in the appendix. Next is a spreadsheet on these tests, giving exception cases and expected result and response message. The pass is given if the response matches the actual response received for that argument:

[Fig Junit.1]

Login Service	Test Case	Input Parameters	Expected Result	Expected Response	Passed Test?
Create Account	Valid Creation	UserAdam, password123	PASS	200 SUCCESS	TRUE
		913213213, jsadliojdasl	PASS	200 SUCCESS	TRUE
		ASdasdaD, aijsaddf	PASS	200 SUCCESS	TRUE
	Invalid Creation	UserAdam, password123	FAIL	Username in use	TRUE
		sda, dgasdasdg	FAIL	Username too short	TRUE
		AdamLand, a	FAIL	Password too short	TRUE
		A, B	FAIL	Username + Password too short	TRUE
		dfasdfsdfdsafdsfdsdsf,fagdfg	FAIL	Username too long	TRUE
		@Adsa@FSA, dsfsa	FAIL	Username contains illegal characters	TRUE
		,jsosasda	FAIL	Must enter a username	TRUE
		asdasp,	FAIL	Must enter a password	TRUE
		,	FAIL	Must enter a username & password	TRUE
Login Account	Valid Login	UserAdam, password123	PASS	200 SUCCESS User Logged In	TRUE
		AdminAdam, adminPassw	PASS	200 SUCCESS Admin Logged In	TRUE
	Invalid Login	UserAdam, password123	FAIL	423 User already logged in.	TRUE
		fadsds, dsfadfsa	FAIL	404 User Not Found	TRUE
		UserAdam, wrongpassword	FAIL	Incorrect Username/Password	TRUE
		UherAdam, password123	FAIL	Incorrect Username/Password	TRUE
		Jsappok,	FAIL	You must enter a password	TRUE
		,dsfadsfds	FAIL	You must enter a username	TRUE
		,	FAIL	You must enter a username & password	TRUE
Connect	Valid Connect	192.128.0.3, 20003	PASS	Connected to server at: /192.158.0.3	TRUE

	Invalid Connect	192.168.0.8, 20003	FAIL	Cannot connect to server.	TRUE
		192.168.0.7, 20004	FAIL	Cannot connect to server.	TRUE
		192.168.0.7, ABCDE	FAIL	Invalid Port Entry	TRUE
		ABCDE, 20003	FAIL	Invalid IP Entry	TRUE
		ABBC, ABBC	FAIL	Invalid Entries.	TRUE
		,	FAIL	You must enter an IP & Port	TRUE
		123.325.324,	FAIL	You must enter a Port	TRUE
		20,003	FAIL	You must enter an IP	TRUE

Room Interactions Unit Tests

For testing the TextRoom implementation, we will also be initially be unit testing the room interactions. Like the Login service tests, these tests will be focused on expected results vs actual result. Using this methodology, we can find preliminary bugs within the code related to room interactions. For the next few tests, another few variables must be initiated on the system; as well as the previous parameters:

Testing Parameters
Open Room is Open, Text Room.
Conclave Room is open, Conference Room
Private Room is closed, Text Room
Locked Room is open, Password: passw123

After the server state has been initialized, the Unit testing can occur. These Unit tests were also done in the order they appear:

[Fig Junit.2]

UserInteraction Service	Test Case	Input Parameters	Expected Result	Expected Response	Passed Test?
UserAdam	joinRoom	Open Room	PASS	You have joined Open Room	TRUE
UserAdam	leaveRoom		PASS	You have left Open Room	TRUE
UserAdam	joinRoom	Locked Room, 2313	FAIL	Incorrect Room Password	TRUE
UserAdam	joinRoom	Locked Room, passw123	PASS	You have joined Locked Room	TRUE

UserAdam	leaveRoom		PASS	You have left Locked Room	TRUE
UserAdam	joinRoom	Closed Room	FAIL	This room is closed	TRUE
UserAdam	joinRoom	Conclave Room	PASS	You have joined Conclave Room	TRUE
UserAdam	postMessage	Hello There!	PASS	[HH-MM] UserAdam: Hello There!	TRUE
UserAdam	postMessage	1@#~+=-_03+]=-[	PASS	[HH-MM] UserAdam: 1@#~+=-_03+]=-[	TRUE
UserAdam	sendPrivateMessage	AdminAdam, heythere	PASS	to [AdminAdam]: heythere	TRUE
AdminAdam	recievePrivateMessage	*Automatically Recieves*	PASS	from [UserAdam]: heythere	TRUE
UserAdam	Export Chatlog		PASS	Conclave_Chatlog[mm-dd]xxx1.txt saved	TRUE
UserAdam	Export Chatlog		PASS	Conclave_Chatlog[mm-dd]xxx1.txt saved	TRUE

Performance and Optimization tests

After verifying that all methods return acceptable results and have passed, we can start to optimise the service. Optimisation is vital for a service like Conclave because the server must respond to multiple requests at the same time, and it must respond quick enough that clients can connect and receive responses rapidly to achieve fluid meetings and near-instant server responses. To test conclave in this respect, both the PerformanceLogger class and the ConclaveLoadTester project will provide interesting results that will allow us to change implementation details. By changing these details we can hope to achieve 2 results: to reduce server-side overheads, such as CPU load and allocated memory used, and also to reduce client-side response times.

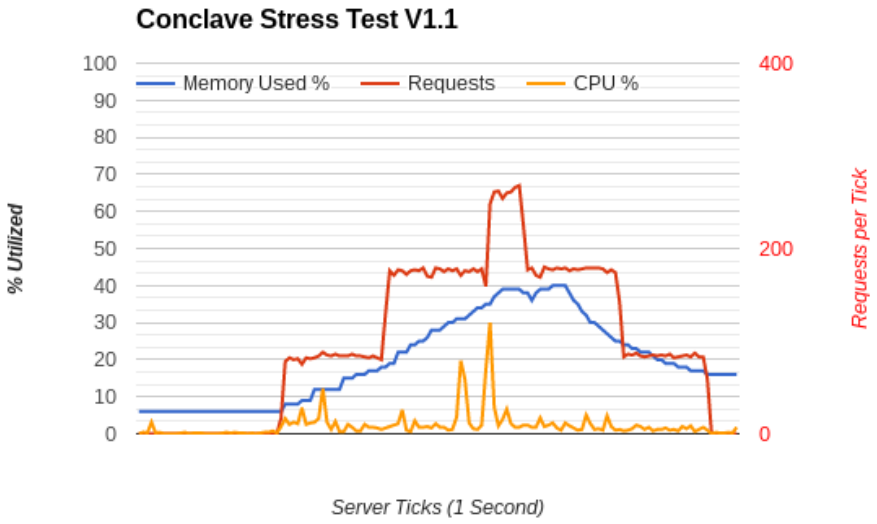
Optimizing ServerController resources

A few performance tests were done on the system in order to meet these results. The first test was a general database request test on the server, with an increasing amount of concurrent client requests to simulate increasing server connection. This is because we should use a requests that utilizes the server data, such as LOGIN request or a SETUP-ACCOUNT request. These types of requests will use JPA to access the database and request account information to validate a number of requirements.

In order to stimulate an actual conclave environment, however, the number of accounts is an important variable as for each request the handler would have to iterate through the collection until it found or didn't find a username. As such, 500 accounts were added to the database for the next cycle of testing. The next graph shows the server state when 3 concurrent load testers on different computers on the network were used to send 15000 LOGIN requests that return a "User not found" response over x server

ticks. The load agents were added at intervals along the server lifecycle, so we can visualise better the effect each new spike would cause:

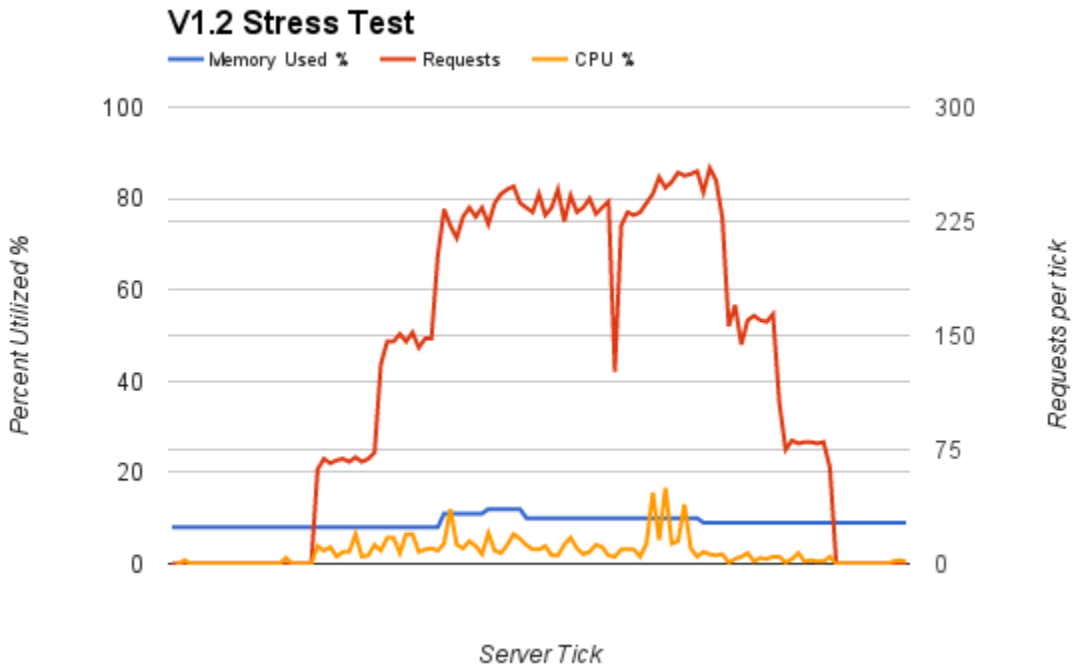
[Fig 4.5]



As you can see from this result the Server was using up to 40% of its allocated memory, and 30% of its CPU on what seems just login requests, I hypothesize that this is because it is holding an incrementing number of concurrent threads, that it stored in a collection, to handle each connection.

A new approach to handling these requests was developed, to incorporate an ExecutorService using a fixed thread pool count. This approach would limit the amount of concurrent threads active, and allow requests to be queued into a pool for execution. This is deployed in a service bean like interaction using a stateless bean pool to handle one-shot requests. This was hypothesised to reduce the memory used, CPU load which would stem from the dramatically reduced thread count. The next graph shows the same test that was done before, using a fixed thread pool count of 10:

[Fig 4.6]



From this graph, we can see that we have dramatically reduced the memory used during the stress test, and even during periods of massive load the memory used never went over 15%. This was believed to because of the dramatically reduced threadcount, and therefore there was no longer multiple ClientHandler objects all competing for resources. We also have reduced the CPU load of the system, as the CPU never goes above 15% utilized, this was proposed to be because of the less initialization and storage of objects and therefore processes were just focused on dealing with the request.

Overall, this new implementation of an ExecutorService was believed to be a success; as it reduced the memory utilized and CPU load by Conclave. Interestingly, although not part of this chart, the average response times for each test didn't reduce by a noticeable amount. (85ms vs 83ms) This was believed to be because of ExecutorService not having true concurrency, and still queues the process each ClientHandler calls; therefore not reducing the effectiveness of the typical dedicated thread for each request. This new implementation also enables the developer to have more control over the threads, and future implementations can implement features such as thread reaping. This is where inactive threads are shutdown and the memory reallocated, or features such as CPU allocation could be implemented; where CPU taxing processes are allocated more CPU time, leading to a faster response time for these processes.

Login Response Time

Although we have so far reduced Server overheads, I'd like to increase the response time on the server. It was noticed early-on that Conclave is fast at responding to requests that access its internal data, such as PING, or an attempt to login when that user is already logged in which both are completed in 13-

16ms, but slow at using the database 83-85ms . So, I was curious to see the effect of using some collection stored in memory to store Account information, and to use this array to validate details. I hypothesize that this will increase the memory used by the server, but also at the same time increase the response time of the server.

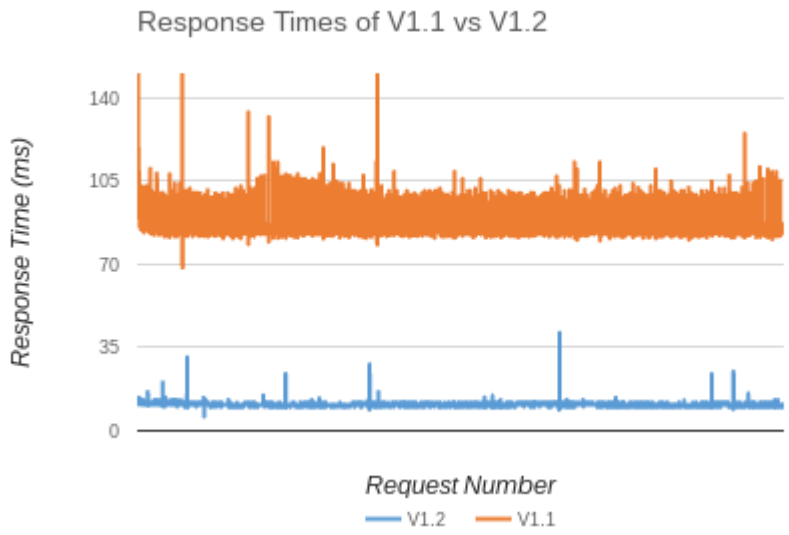
Luckily, JPA has this functionality already available to me, and with a few lines of code I can utilize JPA's "Cache" functionality, which will do the function described above. In order to utilize the cache, i'll have to change my persistence.xml to enable this functionality for a start, add a annotation in my Account object to set the object to be cacheable, and then set the mode of each EntityManager to either "RETRIEVE" or "SET" into the cache, depending of the CRUD operation. Once this had been developed, a 5000 USER NOT FOUND LOGIN request test was done: on the old implementation, named V.1.1, and the new implementation, named V1.2. Their results were logged and the response time average was worked out, producing the following result:

[Fig 4.7]

Version	Average Login Response Time over 5000 requests (ms)
V1.1	85
V.1.2	11

In the interest of further information, here is a chart of the response times logged against request number:

[Fig 4.8]



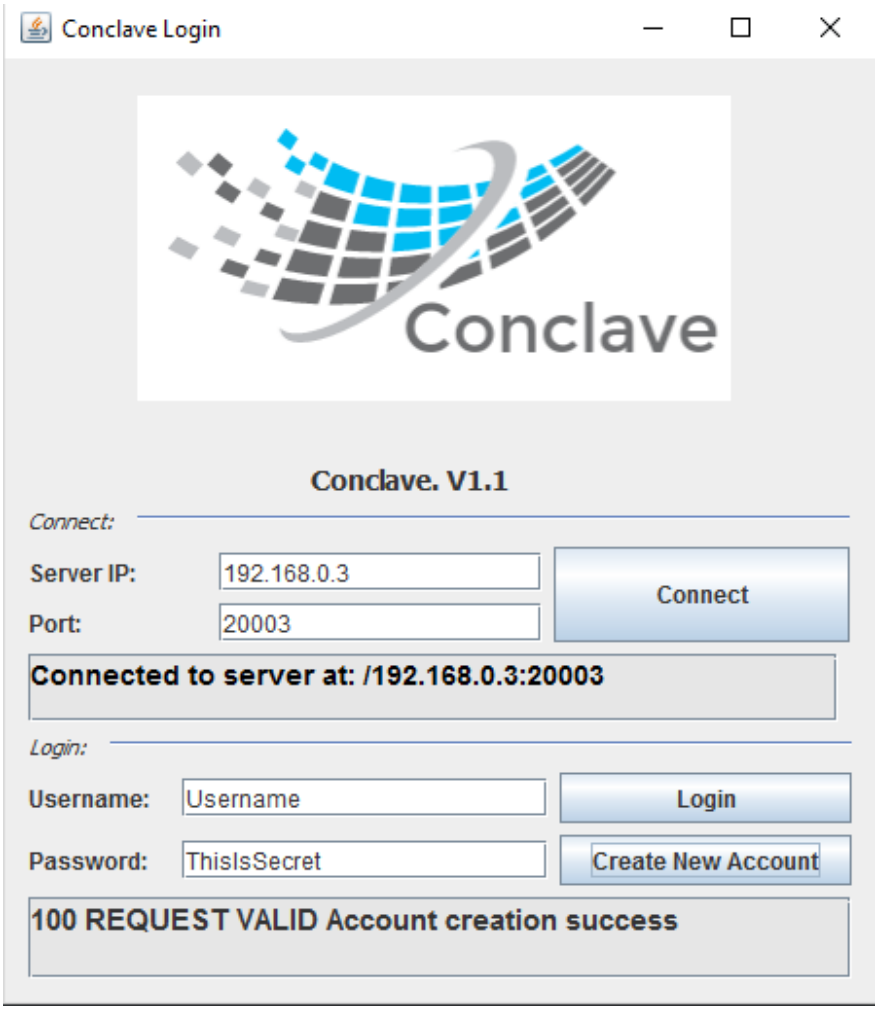
These results were hugely encouraging, and it seems to be the case that we have reduced the response time by 74ms, a huge difference, as well as having the effect of having a more stable response time. I predicted that this optimisation would reduce the response time, but I was not prepared by the massive change using the Cache would have. Thus, in conclusion, this optimization change was a success, and the change would be committed.

5. Second Iteration Development

The second iteration of development will focus mainly on the client side application, and begin to develop the Swing gui in which the UserInterface will be interacted with. The client gui will be built using NetBeans 'Drag n Drop' gui builder. This will simplify development and tedious design orientated code will, mostly, be automatically generated. When designing the interface we must remember that design must adapt to the user's interface, and the possibility for new room types or even new features must be remembered.

5.1 Login GUI

Creating the Login GUI had several requirements, firstly that the user can connect to any conclave server, based on an IP and a Port and then the gui must have some kind of output in which they will be told if the server is live or not. Secondly, users can either create or login an account by entering a Username and a Password; and then they will be informed if it was a success or not; as well as a description as to why. It was decided to include the Conclave logo on the Login GUI, as well as a version number. This decision was made to set apart its design from classic Swing windows, and make it a little bit more unique. The version number was entered to inform users what Client version they were running:
[Fig 5.1]

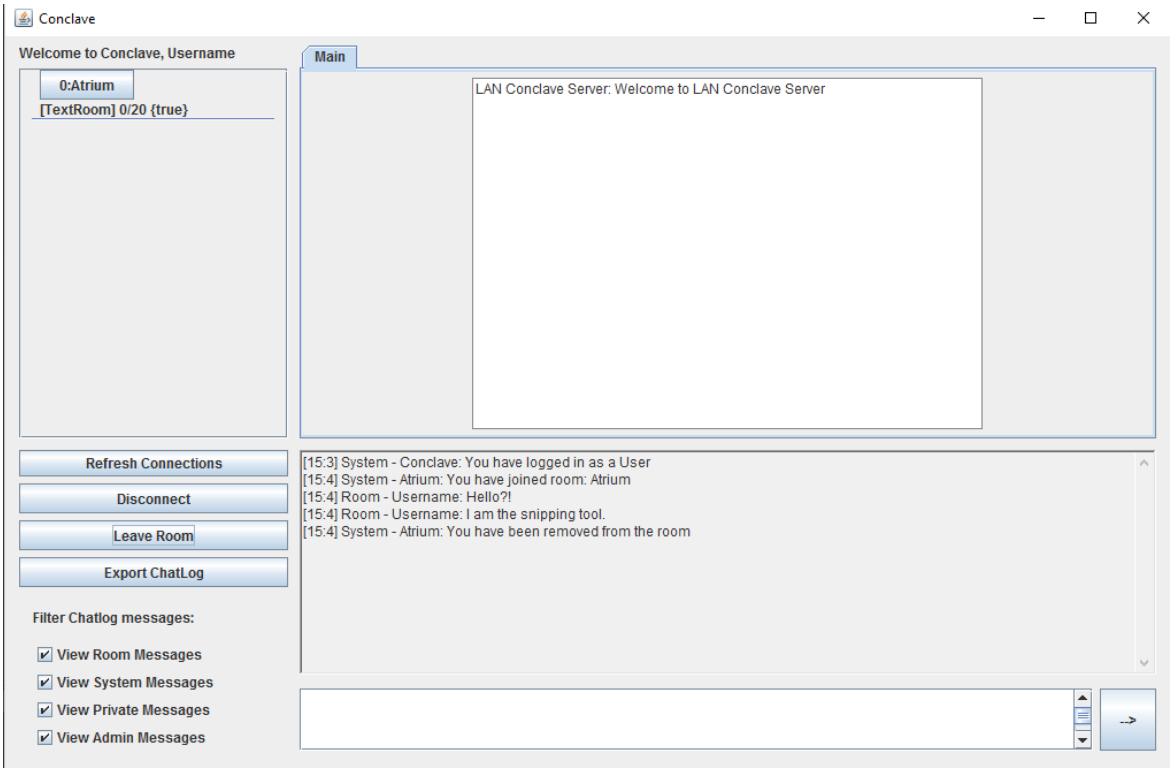


5.2 Client GUI

Designing an aesthetically appeasing interface can be difficult for the developer who primarily focuses on backend code, I found myself in the position of a front-end designer as well as a developer. Thankfully, during the design phase, I designed three UX diagrams which will help me develop my GUI. When developing, I decided to use a main interactable panel which is part of a tabbed panel. This allowed me to easily add new panels depending on what kind of interactions the GUI might need, such as a broadcaster panel or an Admin panel. This also allows new interactions to be built that have not been foreseen in this scope. The side connections log was implemented using a JList, this enabled me to use a list model to ensure connection entries appeared ordered. As the client enters or joins a server, there connection log will change to accommodate either rooms or users to interact with, thus each entry will have a button alongside which the user can use to interact. As well as these navigation implementations, the FrontPage was also implemented using a JList, which will automatically

appear in any roomless user's main tab. My final implementation of this GUI can be found below:

[Fig 5.2]

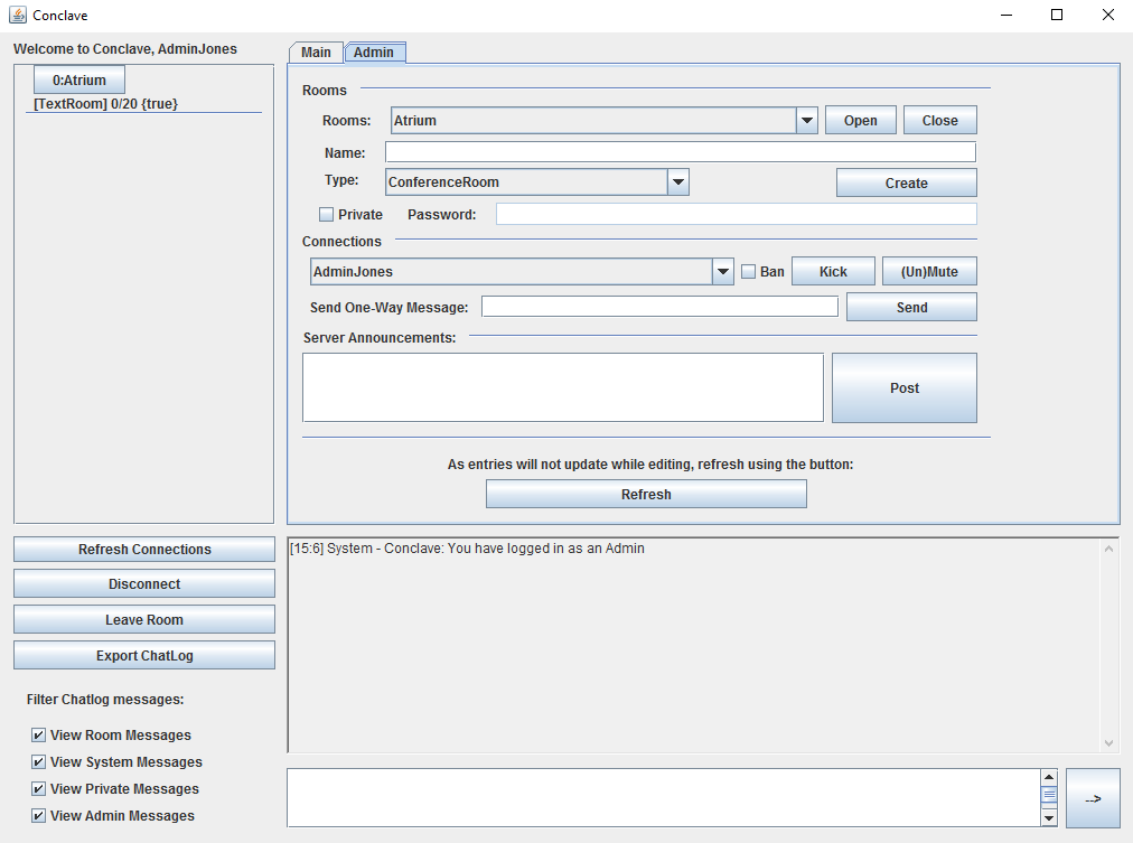


5.3 Administrative Interaction.

Admins on Conclave must have several functionality available to them. They control all connections to a server, modify room visibility and create new rooms, allow or enable certain user interactions like joining a room or even talking in the ChatLog & streaming in a Conference Room. Admin's also have the ability to prevent certain users from even connecting to Conclave. Admins must also be able to post announcements to the Frontpage.

Admin interactions are modelled through the use of an Admin panel, which is added to the tabbed panel. This panel will update, when not displayed, with a list of all connections, persisted rooms and allow specific admin interactions. This tabbed panel has been designed in the UX diagram. The final implementation of this administrative panel can be found below:

[Fig 5.3]



5.4 Look and Feel

In order to allow users to customize what there conclave system will display and interact like, users will have several functionality. Firstly, they will be able to filter certain messages from the ChatLog, displaying only the needed entries that are relevant. Users will then be able to export this ChatLog for use in the future. This will allow users to save certain meeting information and enable them to recall it in the future. Initially, I had planned to allow users to customize their display name, but decided against this as a lot of the server functionality uses usernames, such as the mute/unmute and the server ban list functions.

5.5 Second Iteration Testing and Enhancements

After finishing the second iteration of development, we can begin testing and enhancing the system again, this time the testing will be focused on stateful GUI interactions, such as if the connections log is updated when a player leaves, or that the chatlog is provided with a real time stream of text. We will also be testing the administrative functions of Conclave, ensuring that admin functions produce the intended results without also affecting server overheads too severely. In order to achieve these criteria, we will be performing Unit tests on the admin interface functions, and simultaneously testing administrative functions against server

overheads, such as creating and removing multiple rooms in quick succession, in the style of ConclaveLoadTesting. Unit tests were completed, and the results can be found at:

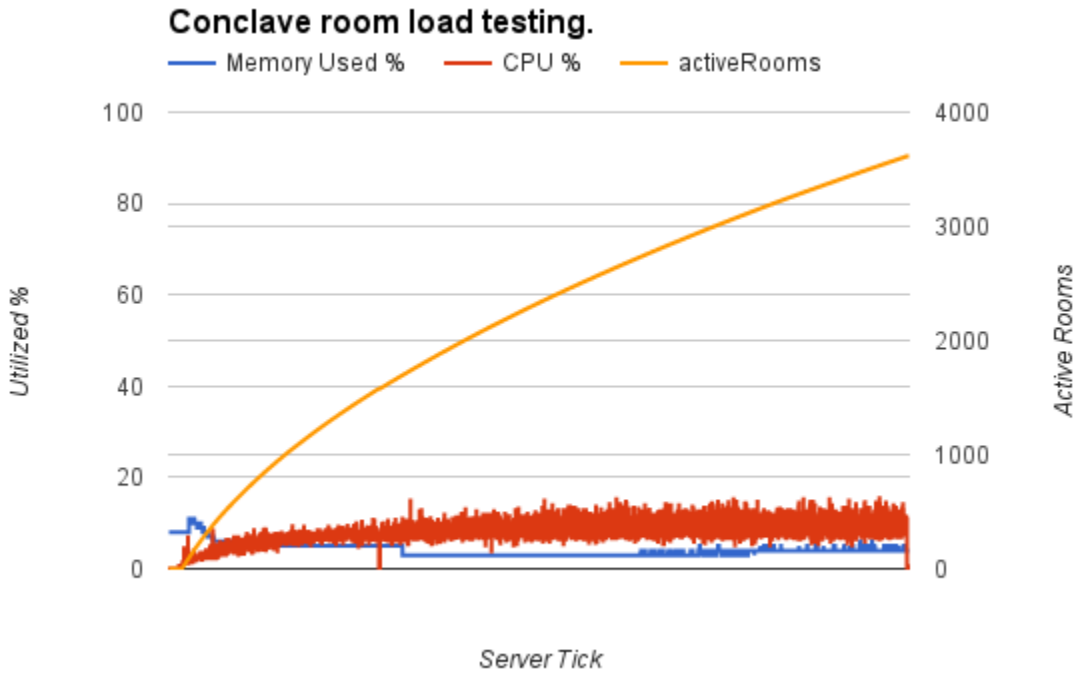
[Fig Junit.3]

AdminInteraction Service	Test Case	Input Parameters	Expected Result	Response message	Passed Test?
AdminAdam	addRoom	NewRoom, 1	PASS	You have created a new TextRoom, NewRoom	TRUE
AdminAdam	addRoom	NewRoom, 1	FAIL	There is already a room with that name.	TRUE
AdminAdam	addRoom	NewRoom, 2	FAIL	There is already a room with that name.	TRUE
AdminAdam	addRoom	NewRoom, 1, roompass11	FAIL	There is already a room with that name.	TRUE
AdminAdam	addRoom	MyRoom, 2, roompass11	PASS	You have created a new ConferenceRoom, MyRoom	TRUE
AdminAdam	openRoom	NewRoom	FAIL	That room is already open.	TRUE
AdminAdam	closeRoom	ConclaveRoom	PASS	ConclaveRoom is now closed.	TRUE
AdminAdam	joinRoom	ConclaveRoom	FAIL	ConclaveRoom is currently closed.	TRUE
AdminAdam	openRoom	ConclaveRoom	PASS	ConclaveRoom is now opened.	TRUE
AdminAdam	joinRoom	ConclaveRoom	PASS	You have joined ConclaveRoom	TRUE
AdminAdam	muteUser	UserAdam	PASS	You have muted UserAdam	TRUE
UserAdam	postMessage	Hi there!	FAIL	You have been muted	TRUE
AdminAdam	unmuteUser	UserAdam	PASS	You have unmuted UserAdam	TRUE
UserAdam	postMessage	HI there!	PASS	[HH-MM] UserAdam: HI there!	TRUE
AdminAdam	kickUser	UserAdam	PASS	You have kicked UserAdam from the room.	TRUE
UserAdam	LeaveRoom	*Automatically Recieves*	PASS	You have been kicked from the room.	TRUE
AdminAdam	adminMessage	UserAdam, "secret msg.."	PASS	to [UserAdam] ADMIN: secret msg..	TRUE
UserAdam	recieveMessage	*Automatically Recieves*	PASS	from [AdminAdam] ADMIN: secret msg..	TRUE

AdminAdam	postUpdate	"This is an update!"	PASS	Frontpage: [AdminAdam]: This is an update!	TRUE
AdminAdam	banUser	UserAdam	PASS	You have banned UserAdam from the server.	TRUE
UserAdam	disconnect	*Automatically Recieves*	PASS	You have been banned, system will exit soon.	TRUE

After ensuring that each interface functionality passes the unit test, a server load test was completed on a standard administrative function of creating a room. This test was developed to find out how the server copes with multiple rooms created and added to the registry in high demand, as such 4000 rooms were added to Conclave and loaded onto the registry, below is a chart that plots server ticks as the x axis vs active rooms per tick and server memory + CPU load percent utilized:

[Fig 5.4]



This result is not conclusive of any real optimization problems, and the only conclusion I can draw from this chart is that the CPU is more active the more rooms there are on the server. This difference is not noticeable, and I couldn't justify any optimization changes for such a small benefit.

6. Third Iteration Development

In this final iteration, we will be implementing key functionality for conclave. We will be developing the Frontpage of the server, where announcements can be posted by Admins for all connected users to view upon login. We will also be developing the Conferencing functionality of Conclave, and this requires us to implement several frameworks that are involved in this interaction.

6.1 Frontpage Development

As the Frontpage is planned to be employing many state interactions, we will used two model classes to model the announcements, one is a ServerFrontpage, which holds a collection of Announcement objects, and the Announcement object. For my implementation, announcement will only have a String message. However, this model design allows different types of Announcements to be created such as a downloadable file announcements for file distribution, hyperlinks or even interactable content like polls or quizzes.
The ServerFrontpage is held by both the client and the server, and the server will call a updateFrontpage() method on each client in its connection collection when a new Announcement is added. When a user logs in, or leaves a room, the server will pass them the entire frontpage array which they set as their local collection

6.2 Conference Room

The conference room implementation is responsible for normal TextRoom interactions, as well as adding functionality of conference based interactions, like starting/stopping broadcast streams and updating users when a stream has started or stopped.

The implementation I developed uses several fields to indicate streamer information: such as IP and Port, Streamer Dimension, Streamer Name. When a client connects to a room that currently has a streamer, they will use this information to build their client broadcast panel and begin streaming. The streaming functionality is based on a peer to peer architecture, and relies on two types of interactions: Streaming, and Listening.

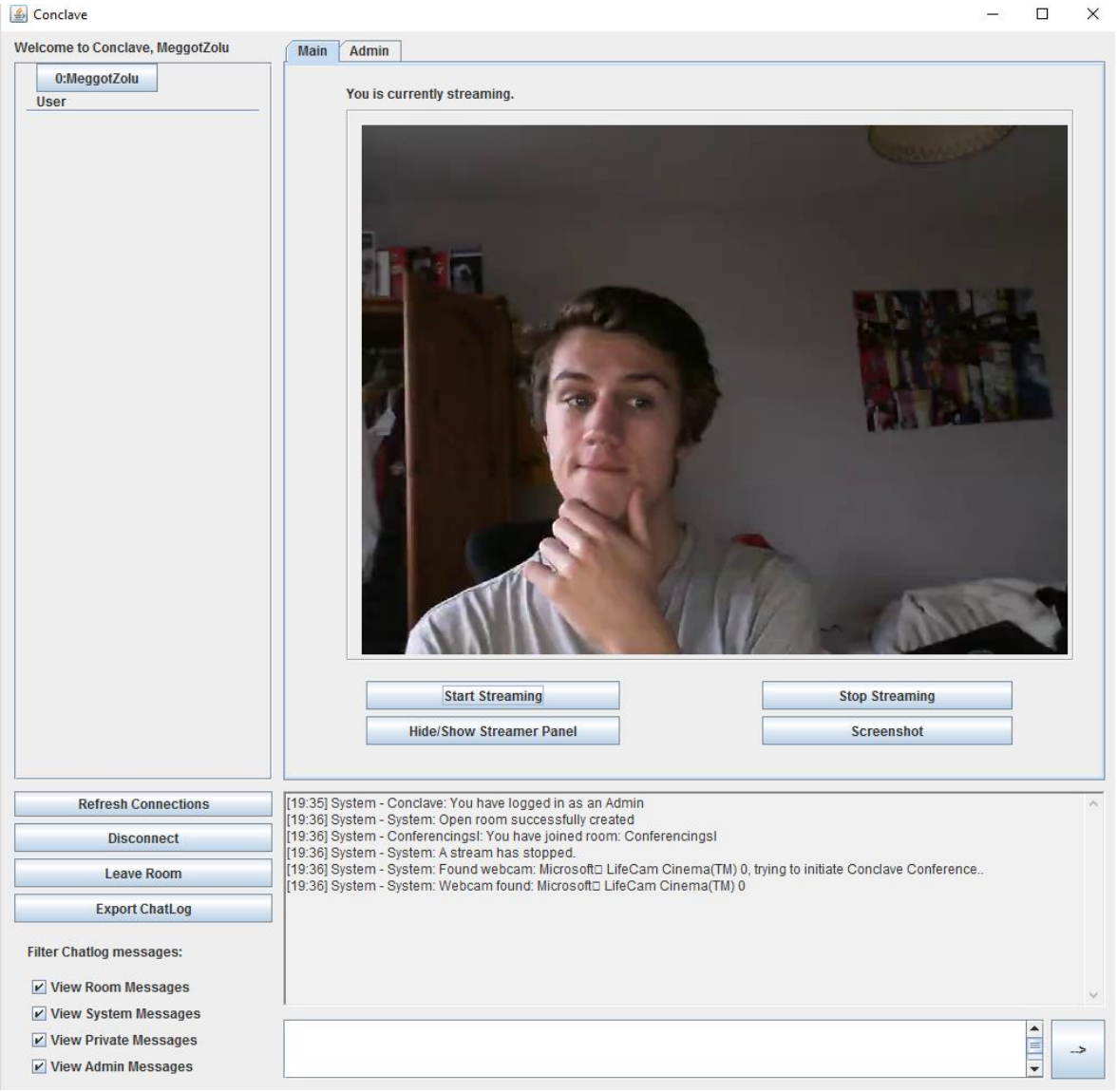
Whilst coming to terms with the frameworks for conclave's Video Conferencing functionality, I found there was many different solutions for the real time transfer of large packets of information: and thus I'll need to choose a specific implementation that will suit my systems needs. A design decision that I made is that video streaming should follow a peer to peer like

architecture, this decision was made to reduce server overheads and reduce unnecessary processing, as in a server/client architecture; frames would have to be transferred across three different networks before reaching the user's display. (The streamers network, to the server network and then finally across to the client's network.) Using a client/client relationship the process is simplified and network responsibility is passed onto the client's system, meaning a ConferenceRoom to use the same overheads as a conventional TextRoom.

To implement this service, I'll be using three different frameworks. For capturing the user's webcam and using the appropriate webcam driver, I will be using Sarxos webcam-capture library which also allows image transformation controls and even motion detection capabilities, but for my purposes I will only be using it to capture webcam and get BufferedImages for each frame. After capturing the frame, I used the Xuggler library to convert this BufferedImage to h.264 video format and allow me to serialize it for usage in transport. To transport frames and ensure each frame is updated and then passed onto the room participants, I will be using Netty IO framework with handlers to repaint a display bean: after Xuggler has decoded the frame, and listener/broadcasting agents to initialize and teardown connections. To utilise this service, the user will consult the conference room to see if there is an active streamer, if there is one then the room will provide the user with the streamers IP and Socket for them to make a Netty channel connection to and start receiving frames from them, if there isn't then a user can choose to start streaming, which will then call an update all on connected users and they will start receiving frames, as well as store the streamers ip and port onto the room for future connections to use.

After this has been implemented, it was needed to design a Conferencing view that would be able to do all the needed functions for a streamer/listener such as starting/stopping the stream, screenshotting the current stream, and hiding or showing the panel. I developed a "Broadcast Panel" that achieved these functions through the usage of JButtons:

[Fig 6.1]



6.3 Third Iteration Testing

During this iteration of development, the most noticeable implementations were the Frontpage functionality, and the ability for users to stream and listen to webcams. In order to verify that these implementations produce the intended results, a unit test was constructed for these functionality. As usual, a server state was initiated and the following unit tests were performed on the server and on the clients:

[Fig Junit.3]

UserAdam	Start Broadcasting.		PASS	UserAdam is now broadcasting!	TRUE
AdminAdam	Start Viewing	*Automatically Recieves*	PASS	You are viewing UserAdam's stream	TRUE

AdminAdam	Start Broadcasting.		FAIL	A user is already streaming.	TRUE
AdminAdam	Hide Stream		PASS	Stream is hidden	TRUE
AdminAdam	Screenshot		PASS	Conclave_Screenshotxxx1.png saved	TRUE
AdminAdam	Screenshot		PASS	Conclave_Screenshotxxx2.png saved	TRUE
UserAdam	Stop Broadcasting		PASS	UserAdam has stopped broadcasting.	TRUE
AdminAdam	Stop Viewing	*Automatically Recieves*	PASS	You are no longer viewing a broadcaster.	TRUE
AdminAdam	Start Broadcasting.		PASS	AdminAdam is now broadcasting!	TRUE
UserAdam	Start Viewing	*Automatically Recieves*	PASS	You are now viewing AdminAdam's stream	TRUE
AdminAdam	LeaveRoom		PASS	AdminAdam has stopped broadcasting, You have left the room	TRUE
UserAdam	Stop Viewing	*Automatically Recieves*	PASS	You are no longer viewing a broadcaster.	TRUE

7. Project Evaluation

In the evaluation I will be discussing how my implementation has succeeded in achieving my requirements set out at the start of the report, and it shall detail any changes I have made to the system, such as design oversights I might have made, problems that I have encountered and solved as well as problems that I cannot solve in the allocated time. During the evaluation we will be referring to main development iterations and how successful I feel each iteration was in terms of achieving my requirements as well as in terms of software engineering practices.

7.1 Requirement Implementation evaluation

How has conclave achieved certain functionality? Why were these implementations chosen to fulfill these requirements? We will be digesting each requirement and discussing these questions:

7.1.1 Functional Requirements

- *Many types of ConclaveRooms can be hosted on a server and any users connected must be constantly informed of room status changes: such as number of users in a room vs maximum capacity, open status, room type and room name.*

Developed in Iteration 1. This was developed in the form of a ConnectionsLog. A connections log allowed users not in a room to view all the available rooms in a consistent form, detailing relevant details. This ConnectionLog served as an interactable element as well, as each entry was built with a button that allowed users to interact with these rooms and join them. To keep design consistent and reuse functionality, this ConnectionLog also allowed users to view all users connected in a room, and interact with them and send them private messages. This design was chosen to allow new types of connections to be intuitively displayed, as well as having reduced overheads as my implementation means the ConnectionsLog only provides basic Strings such as Connection name, and Desc. The client application then decides how to display this.

- *Clients should connect and start to send login information securely and be protected from possible man in the middle security (MITM) attacks such as packet sniffers or spoofing attacks.*

Developed in Iteration 1. This was implemented in the form of a AES encryption key stored on the clients classpath directory, that when loaded from; encrypted all packet traffic sent between client/server. The server would select the appropriate key to decrypt and parse this response, before encrypting and sending over the response.

- *For this implementation of Conclave, there must be two types of rooms implemented: a standard TextRoom, with just ChatLog functionality, and a Conference Room which implements the ability for one user at a time to stream their webcam to all connected and viewing users in that room, as well as the ChatLog.*

Developed in Iteration 2 and Iteration 3. This was implemented in the form of an interface object, ConclaveRoom, which defined the basic functionality that every room would use, like: users joining, leaving, updating client states and posting chat log messages and allowed this basic interactions. Rooms that provided extra functionality on top of this design, such as a ConferenceRoom, should extend this TextRoom.

The conference room functionality was implemented using several frameworks to define this new peer to peer interaction between streamer and listener, this was detailed in iteration 3. This design was chosen to allow for rooms that haven't been foreseen by this scope to be developed later on with ease, and to be used by the system in the normal ways.

- *When joining a server, all users will be presented with a "Frontpage" which is always visible to users that are not currently in a room, and will display all server announcements when they appear. Admins will post announcements.*

Developed in Iteration 3. This was implemented in the form of a stateful model class of ServerFrontpage, which is owned by a ServerController and interacted via an AdminInterface to

add announcements. This frontpage is held by every client too, and updates are then passed to each interface object which then are stored in their own model class. This was more keenly detailed in iteration 3. This type of implementation was chosen to allow ease of use, and make sure that each client receives an updated front page on joining a server, and receive new updates whether they are in a room or not.

- *Users can join any open room, and if that room has a password; be presented with some kind of input service.*

Developed in Iteration 1. This was implemented in part by the interact functionality of a ConnectionLog, and when users select a room to join the client side implementation will check to see if the room is a private room, and if it: the client will prompt an appropriate input window for password capture.
This type of design was implemented with the idea of possibly multiple types of client applications developed after release, and how the prompt is given is determined by the client rather than enforced by the server. The server will simply provide the response.

- *Users can send private messages to any other user that is in their room, these are only seen by the recipient and the sender.*

Developed in Iteration 2. This uses the ConnectionLog implementation to interact with connections. An appropriate input form is prompted which will allow clients to send a private message to the connection. The current room both users are in will then send an update to the recipient.

- *Once inside of any room, users that are not muted can post in the room chatlog, and all users inside the room will receive that message in real time.*

Developed in Iteration 1. The chatlog functionality was implemented through the use of a Message model object, this object will contain the type of message in the form of an enum class, be it System, Room, Private or Admin. This model object will also contain the sender username and the recipient username. The active room that the user is in will then pass the message to the correct recipients from this object. The room object also stores a list of muted usernames, and will prevent these users from posting in the room as well as preventing them from streaming in a ConferenceRoom. This design was chosen as it allows for Messages to be stored in a serialized form and displayed as needed without additional classes being necessary for different types of message. It also allows the client to display certain message types differently as it needs.

- *Admins will be given the correct interface upon login, which will allow them to perform operations on rooms such as: Add/Remove rooms from the server, Open/Close rooms to*

new connections, mute users from chatting and streaming their webcam, kick users from rooms and also ban them from the server.

Developed in Iteration 2. This was implemented in the form of an administrative interface implementation that implements this functionality. The servercontroller class will create an appropriate interface object, based on if the username is in the "Admins" arraylist. The controller uses a RoomManager object to do most of these admin interactions, and as this uses a singleton pattern each admin will use the same RoomManager in their interactions, preventing duplicate room listings. This object extends the UserInterfaceImpl object, allowing it to be passed to a client in the same way. This implementation was chosen to allow the client to call a getType() request on the object and if it returns 2, then the object is an Administrative implementation, and the client will then build an appropriate panel that will model these interactions. It was also chosen as this allows multiple types of users types to be created in the future.

- *The system will handle disconnecting clients gracefully, and ensure that all clients the server shows are actual connected, and interacting clients.*

Developed in Iteration 1 This was addressed in the form of a Connection Manager thread which runs upon server startup. This thread will iterate through all connections and call a "isConnected()" method on them, if the method throws a RemoteException then the user has terminated their connection unusually, or been disconnected. If the method returns a false, then the user has called a Disconnect interaction or has been banned from the server; and they are subsequently removed from the connections array and their departure logged. This implementation will ensure that the connections information maintained by the server is correct and each user is interacting.

7.1.2 Quality Requirements

- *Private Rooms and Users are persisted into a Database, with their passwords stored securely in a hash code, combined with an added protection of a random salt.*

Developed in Iteration 1. This was achieved through the usage of manager classes taking advantage of an EntityManager to persist into a derby database, all password handling was performed by a SecurityHandler which performed these encryption techniques. Salts were first generated using a RandomKey generator, then hashes were created using a SHA-256 hashing algorithm. Validation of input are performed by the RoomManager and AccountManager.

- *Users should be able to export their ChatLog at anytime, which will save it to their local ConclaveClient directory in the form of a readable .txt file.*

Developed in Iteration 2. This was implemented in the form of a ChatLog method which returns all Messages in a LinkedList, the Client will specify the last line it has received to the UserInterfaceImpl object which will return a list of all messages that has occured soon. This chat log is printed out in a printwriter to a txt file to a local file specifying the roomname, and system date, as well as an incrementing static integer as the filename.

- *Users connected to a ConferenceRoom can take screenshots of the current active stream at anytime, this will save it also to their local ConclaveClient directory in the form of a .png.*

Developed in Iteration 3. This was implemented by using an ImageWriter on the current BufferedImage which saves it to the Client folder using a hash code as a filename. An output was written to notify that the screenshot was saved.

- *Users should be able to control their Chat Log message display entries and choose to filter out certain message types such as Admin, Private, Room, and System.*

Developed in Iteration 2 and 3. This was implemented using a client side array containing a list of the types of messages that the chatlog should display, the GUI will provide the ability to add/remove types from this arraylist. When a change to the filter settings occur, the client will reset its last message line received to 0, forcing a total update from the userinterface object of all message; which it will compare against its type array.

- *Users should be able to refresh their connections with an input, it should update automatically but this will allow Users to verify their own state whenever.*

Implemented in Iteration 2. This was developed in the form of a button provided by the GUI which will call a refresh connections log, updating any connections and chatlog updates immediately rather than when the next client tick update happens

- *All administrative interactions with the server are to be logged using a Logger implementation, which will log useful information such as usernames and parameters. Logger will also log users logging in and out and any exceptions the server throws.*

Implemented in Iteration 2,3 This was developed using Java's Logger functionality, using appropriate Logger level to indicate the logging type, exceptions were logged using Level.SEVERE, whereas informative logs such as administrative interactions, users joining/leaving a room and connections made to the server, were logged using Level.INFO.

- *All server overheads will be logged into a suitable .csv format, writing such details like CPU performance, memory overheads and thread count, along with server states like*

active room, logged users, and anonymous connections (These are connections that have not logged in yet).

Implemented in Iteration 1. This tool proved extremely useful for testing, and produced accurate results in terms of optimization changes. This allowed me to gauge how successful some optimization decisions actually were, and also allowed me to identify problems that may be occurring such as memory leaks, or CPU inefficiencies.

- *Admins should be able to send one-way messages to any connected user of the server, this is intended to provide a warning or to inform users of a room password.*

Implemented in Iteration 2. This was implemented using the Admin interactions controlling the ServerController to send an update chatlog call to the correct connection in its collection. This was implemented to allow administrators to send warning messages, or notify people of a password.

7.2 Development notes and iteration evaluation

Throughout development a lot of methodology was taken, and in this section i'll be detailing what these approaches were, both in times of finding appropriate resources for your design and also controlling development flow through the use of version control and scheduling information.

Development

Development schedule was scheduled by a Gantt chart, and although this was used as an estimate of how long each task would take, it often was not adhered to. I feel this is because I underestimated how much time it would take to implement the ConferenceRoom functionality, I hadn't originally anticipated that I would end up having to use 3 different frameworks during its implementation, but after looking at several implementations I decided that the current "all-in-one" frameworks provided too much functionality for my use, and if I decided on using them would end up with much more overheads and less fit for my lightweight purposes.

For my development research I looked at code on Sarxos webcam examples, and found a decent netty implementation that involved IP cameras across a network. This example allowed me to develop a Swing based peer to peer video that featured only 1 way conferencing, although my implementation does allow for multiple users I found I ran out of development time to continue development in that direction, but perhaps this will be developed after the report's writing.

In order to aid development, classes were annotated using Javadoc standard, outlining parameters and behaviours of each methods as well as a description on each class detailing its responsibilities and behaviour. This helped me use NetBeans IDE tools to view important implementation behaviour as development occurred.

Version control was managed by a subversion account hosted at assembla.com (https://www.assembla.com/spaces/conclave/subversion/source), and some commits provided useful information in what was achieved in each update, in some cases commits were under commented, as it was hard to remember exactly what was done since last commit; so for future projects I'll be sure to keep a jotting of tasks accomplished in order to produce a more thorough commit message.

Below is the commit log, with dated commits and committer, with a supplementary commit message, and on the right being a version number. Bear in mind that some of these commits were committed to either 4 of project directories, ConclaveInterfaces, Conclave, ConclaveClient or ConclaveLoadTester: [Fig 7.1]

Friday, April 08

April 8th 2016, 19:14:58

Bradley_Williams

did some cleanup, made the project jar friendly.

32

April 8th 2016, 19:13:36

Bradley_Williams

Cleaned up the admin panel, added .connect() so Conclave will disconnect users that login but don't have a GUI. Fixed a few bugs and provided more input dialogs to the user whenever a transaction occurs.

31

April 8th 2016, 19:12:25

Bradley_Williams

cleaned up the Performance logger, optimized JPA functions by using CACHED copy instead of accessing from database, whenever retrieve methods are called. Cleaned up the userinterface and servercontroller.

30

Wednesday, April 06

April 6th 2016, 18:09:12

Bradley_Williams

cleaned up code, allows the databse tables to be created upon connecting.

29

Thursday, March 31

March 31st 2016, 14:57:02

Bradley_Williams

Commented code, cleaned up the kick/leave functionality and fixed a bug involving kicked players never being removed from connectionslog.

28

March 31st 2016, 14:56:48

Bradley_Williams

Commented code, cleaned up the kick/leave functionality and fixed a bug involving kicked players never being removed from connectionslog.

27

Monday, March 28

Bradley_Williams

cleaned up the ServerController, also made the RMI system host ip registry be setup to localhost rather than hard coded ip.

26

Bradley_Williams

March 28th 2016, 19:45:14

Code cleanup, provided better dialog controls on broadcasting panel, fixed the bug which caused webcams to not be updated when a new user streams, and the bug which hides streaming name.

24

Bradley_Williams

March 28th 2016, 19:43:19

This will load test the conclave server, and the gui enables people to stack multiple socket requests, with an iteration count.

more

21

Bradley_Williams

March 26th 2016, 20:28:50

Codecleanup. Changed the LoginGUI to make it more visually appeasing, also reshuffled the broadcast panel. stopped video panel flickering by removing validate() calls in frame update. Also added proper libs instead of importing directly from ConclaveServer, allowing a distribution jar which runs in any JVM.

20

Thursday, March 17

Bradley_Williams

March 17th 2016, 19:47:36

Final commit, concurrency safe. Added screenshot, export txt, moved administrative interactions to a dedicated Panel to clear up code.

19

Bradley_Williams

March 17th 2016, 19:46:39

Final commit, concurrency safe.

18

Wednesday, March 16

Bradley_Williams

March 16th 2016, 21:25:06

Added Logger all over Room Controls, Admin Interactions, user interactions that would be watched in the future. Added more possible responses to the Login service to include logged in, banned and such. Also changed the way Server stores connections, and will routinely check to see if users are still connected to a server and remove them from its collection. Added this method call on some RemoteException exceptions to make sure we dont have offline players in this arraylist.

17

Bradley_Williams

March 16th 2016, 21:22:20

Added a Broadcast Panel which uses the VideoPanel, this makes the whole broadcasting interaction more O-O and allows me to add stream handler buttons to control specific "Broadcasting" domain interactions such as a Hide Stream, Screenshot, Stop Streaming, Start Streaming buttons. Also updated the Login Controller and GUI to display Already Logged in outputs, and Banned User outputs. Cleaned up SwingGUI.

16

Saturday, March 12

Bradley_Williams

March 11th 2016, 23:30:10

Added functions directly to the SwingGUI to model UI of the AdminInterface instance, these controller methods are detailed in the Conclave commit previously. Also added new streaming interactions and cleared up the webcam methods. Added multiple new classes dedicated to the conferencing ability of Client. The client/client like interaction of conferencing rooms will hopefully reduce bandwidth and server overheads, and pave the way for future Conclave interactions. These interactions use new classes to handle encoding and decoding, both from frame formats and h264 livestream example. These encoding classes were taken (and then modified for purpose) from Sarxos-Webcam live streaming example, and in turn use a Xuggler framework.

15

Bradley_Williams

March 11th 2016, 23:24:49

Added a new Interface, AdminInterface, which is then implemented by AdminInterfaceImpl. This class handles all the CRUD operations on a active server both in terms of room management, user management and "ServerFrontpage" management. ServerFrontpage is used to display announcements and give users information upon login, later these announcements can be extended to include pictures, files, weblinks and more. Announcements are not persisted, although this may be implemented further down.

more

14

Wednesday, March 02

Bradley_Williams

March 2nd 2016, 20:25:13

Changed the main panel to a JTabbed pane

13

Wednesday, March 02

Bradley_Williams

March 2nd 2016, 18:14:56

Updated SwingGUI features to include streamable rooms

12

Bradley_Williams

March 2nd 2016, 18:14:00

Updated features.

11

Tuesday, March 01

bradley_williams

March 1st 2016, 16:55:33

Updated the SwingGUI to include filtermessage checkboxes which will ultimatly allow the user to filter chatlog messages from their final display.

10

bradley_williams

March 1st 2016, 16:53:42

Updated the ConnectionHandler to use try/catch connection exceptions on server create, and setup connection requests to handle JPA closages.

more

9

Bradley_Williams

March 1st 2016, 14:26:34

Moved the buttons up a few pixels.

8

Bradley_Williams

March 1st 2016, 14:24:28

rearranged the chatGUI

7

Monday, February 29

Bradley_Williams

February 29th 2016, 21:39:40

Deleted misc files

6

Bradley_Williams

February 29th 2016, 21:34:20

Added Client code, implements the User interface, Admin interface to come.

5

Bradley_Williams

February 29th 2016, 21:30:13

Added initial code, implements 30% of features.

3

7.3 Future Improvements

Conclave has many areas which more functionality could be added to, and as the design is as non-coupled as it could be, new classes and functionality can be added on top of existing model classes or controller classes without affecting previous functionality. Such improvements such as:

- The ability for multiple users to stream their webcams to a room participants at the same time. This would be implemented in the form of a new type of room.
- The ability for users to also stream their microphone to users. This was disappointingly not managed to be implemented into Conclave for release, although the implementation was researched I forgot to plan for this to be developed alongside the webcam and ended up with no more time for development. This would be developed using a TargetLineData implementation, and another netty channel to handle audio frames.

- File sharing functionality, both in announcements and in ChatLog Messages. This would be implemented through the use of FileOutputStreams and FileInputStreams using sockets.
- Screen sharing functionality, to let users stream their current active window and perhaps a screen partition to all active room users. This would be implemented using Java.awt robot's functionality on top of Java's GraphicsEnvironment to capture screenshots at certain fps intervals.
- Friendslist and friend invites, with conclave providing login/logout notifications and the ability to always send private messages to them. This would be implemented using a handler that ran whenever a user logged in and logged out that would send them a private message, and for the view a new panel would be implemented on client side.
- Email invitations, this would allow administrators to send emails to email addresses providing an interactable link which they could interact with, launching a conclave session or prompting a download of the client program. This would be implemented through the use of java servlets and an email server.
- Polls and quizzes that run in the frontpage, allowing the server to post managerial decisions and gauge a response. This would be implemented in the form of an AnnouncementBehaviour interface class that implements a strategy pattern.
- Room behaviour api, to allow developers to develop plugins that give rooms behaviours not common to either room implementations such as games, anonymous polls or version control API interactions. This would be done by using a RoomBehaviour interface class that implements a strategy pattern.
- Conclave over the internet. This would be done using a SSH tunnel, I decided this would be the approach to take, as RMI over the internet would require both sides to port forward their connections as well as add exception rules on their firewalls. Also SSH connections are notoriously secure and wouldn't pose a threat as much as RMI over the internet would.

8. Conclusion

In conclusion I feel that I have achieved all the functionality defined in my requirements and implemented a project that adheres to certain design principles. In terms of my time management schedule, I feel that I started in good time which allowed me to develop alongside tests throughout each iteration which ultimately aided my testing sections by producing quantifiable results of each decision and problem encountered. Unit testing produced valuable implementation testing results that allowed me to see if code I had written was actually achieving my functionality requirements.

This type of development, XP, proved to be a brilliant environment that allowed me to take a laid back reflection period after each iteration of development, allowing me to conceive of possible performance drawbacks and error-cases each implementation decision might've made, and then to test these theories.

I might've achieved better results, however, If I had taken a version control approach earlier, as well as bulking out my commit messages with a more thorough detailing of the tasks accomplished. I feel this would be the case because more detailed and uniform commit messages would have allowed me identify each iteration in terms of milestone commits, and allow me to see more accurately where on the iteration schedule my development was currently at.

I feel that the design of my UI could have been executed better, and by the usage of beans I could have created a more consistent and reliable view implementation. Often I found strange bugs within the broadcasting display that caused some parts of the windows to be displayed in poor time, freezing the system. I attributed this to Swing and its tendency to overpaint components.

I also feel that I could've developed more functionality to my system, the ability for audio streaming and file sharing is something that wouldn't of taken much more development but in the end i decided to focus on testing and optimization of the features I did implement. Although the two features named above would not much work in reality to implement, I also decided that stretching out development would cause what could be considered an anti-pattern of endless recursive functionality that in the end affects negatively performance milestones and optimization features.

9. References

8x8 Meetings. (n.d.). Retrieved from https://www.8x8.com/web-conferencing.
Adobe Connect. (n.d.). Retrieved from http://www.adobe.com/uk/products/adobeconnect/meetings.html.
ArgoUML - UseCase v2. (n.d.). Retrieved from http://argouml.tigris.org/.
Darcula - Netbeans Theme Shown. (n.d.). Retrieved from http://plugins.netbeans.org/plugin/62424/darcula-laf-for-netbeans.
DrawIO - UseCase v1. (n.d.). Retrieved from https://www.draw.io.
EasyUML Netbeans Plugin. (n.d.). Retrieved from http://plugins.netbeans.org/plugin/55435/easyuml.
Moqups - UX Diagrams. (n.d.). Retrieved from https://moqups.com/.
NettyIO. (n.d.). Retrieved from http://netty.io/.
Sarxos WebcamCapture Framework. (n.d.). Retrieved from https://github.com/sarxos/webcam-capture.
Visual Paradigm - Sequence Diagrams. (n.d.). Retrieved from https://www.visual-paradigm.com/.
WebEx. (n.d.). Retrieved from https://www.webex.co.uk/.
Xuggler. (n.d.). Retrieved from http://www.xuggle.com/xuggler/.

10. Appendix

Source Code Listing:

Interface Project

IAdminInterface class:

```
package conclaveinterfaces;

import java.rmi.RemoteException;
import java.util.List;
import model.Message;

/**
 *
 * @author BradleyW
 */
public interface IAdminInterface extends IUserInterface{
   public void addRoom(String roomname, int roomType) throws RemoteException;
   public void addRoom(String roomname, int roomType, String roompassword) throws RemoteException;
   public void removeRoom(String roomname) throws RemoteException;
   public List<String> getRoomNames() throws RemoteException;
   public void kickUser(String username, boolean banned) throws RemoteException;
   public void closeRoom(String roomname) throws RemoteException;
   public void openRoom(String roomname) throws RemoteException;
   public void censorUser(String username) throws RemoteException;
   public void uncensorUser(String username) throws RemoteException;
   public boolean isMuted(String username) throws RemoteException;
   public List<String> getSupportedRoomTypes() throws RemoteException;
   public List<String> getAllConnectedUsernames() throws RemoteException;
   public void sendAdminMessage(Message message, String username) throws RemoteException;
   public void postAnnouncment(String msg) throws RemoteException;
}
```

IConclaveRoom class:

```
package conclaveinterfaces;

import java.rmi.Remote;
import java.rmi.RemoteException;
import model.ConnectionsLog;
import model.Message;

/**
 *
 * @author BradleyW
 */
public interface IConclaveRoom extends Remote{

   public void stopRoom() throws RemoteException;
   public void addUser(String userId, IUserInterface user) throws RemoteException;
   public void removeUser(String userId) throws RemoteException;
   public boolean hasUser(String username) throws RemoteException;
   public void postMessage(Message message) throws RemoteException;
   public void setLimit(int limit) throws RemoteException;
```

```
    public void updateAllClientsChatlog(Message msg) throws RemoteException;
    public String getRoomName() throws RemoteException;
    public int getRoomLimit() throws RemoteException;
    public boolean isOnline() throws RemoteException;
    public String getInfo() throws RemoteException;
    public int getType() throws RemoteException;
    public void updateAllClientsConnections() throws RemoteException;
    public ConnectionsLog getAllConnections() throws RemoteException;
    public void whisper(Message msg) throws RemoteException;
    public void closeRoom() throws RemoteException;
    public void openRoom() throws RemoteException;
    public void addCensoredUser(String username) throws RemoteException;
    public void uncensorUser(String username) throws RemoteException;
    public void kickUser(String username) throws RemoteException;
}
```

IConferenceRoom class:

```
package conclaveinterfaces;

import java.awt.Dimension;
import java.net.InetSocketAddress;
import java.rmi.RemoteException;

/**
 *
 * @author BradleyW
 */
public interface IConferenceRoom extends IConclaveRoom {

    public Dimension getDimension() throws RemoteException;
    public boolean isStreaming() throws RemoteException;
    public void startBroadcasting(String username, InetSocketAddress networkloc, Dimension d) throws RemoteException;
    public void stopBroadcasting(String streamerName) throws RemoteException;
    public InetSocketAddress getStreamerIp() throws RemoteException;
    public String getStreamerName() throws RemoteException;
}
```

IUserInterface class:

```
package conclaveinterfaces;

import java.awt.Dimension;
import java.net.InetSocketAddress;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.LinkedList;
import java.util.List;
import model.Announcement;
import model.ConnectionsLog;
import model.Message;

/**
 *
 * @author BradleyW
 */
public interface IUserInterface extends Remote {

    public boolean inRoom() throws RemoteException;
```

```
public boolean joinRoom(String roomName) throws RemoteException;

public boolean joinRoom(String entryName, String password) throws RemoteException;

public boolean hasPassword(String entryname) throws RemoteException;

public void leaveRoom() throws RemoteException;

public String exportChatLog() throws RemoteException;

public void postMessage(String message) throws RemoteException;

public void updateChatLog(Message message) throws RemoteException; //Called by ConclaveRoom

public void updateConnections(ConnectionsLog newLog) throws RemoteException; //Called

public ConnectionsLog viewAllConnections() throws RemoteException;

public void sendPrivateMessage(String message, String recipientID) throws RemoteException;

public void recievePrivateMessage(Message message) throws RemoteException;

public void leaveServer() throws RemoteException;

public boolean isConnected() throws RemoteException;

public void connect() throws RemoteException;

public void disconnect() throws RemoteException;

public boolean hasConnectionsUpdated() throws RemoteException;

public String getUsername() throws RemoteException;

public String getActiveRoomName() throws RemoteException;

public int getRoomType() throws RemoteException;

public LinkedList<Message> getChatlogUpdates(int lstMsgRecieved) throws RemoteException;

public int getLastMessageLine() throws RemoteException;

public int getType() throws RemoteException;

public List<Announcement> getFrontpage() throws RemoteException;

public void setFrontpage(List<Announcement> frontPage) throws RemoteException;

public void updateFrontpage(String username, String announcment) throws RemoteException;

public void updateStreamer() throws RemoteException;

public boolean hasFrontpageUpdated() throws RemoteException;

public Dimension getConferenceDimension() throws RemoteException;
```

```
    public String getStreamerName() throws RemoteException;

    public void stopBroadcasting() throws RemoteException;

    public boolean isConferenceStreaming() throws RemoteException;

    public InetSocketAddress getStreamerLocation() throws RemoteException;

    public void broadcastToConference(InetSocketAddress loc, Dimension d) throws RemoteException;

    public boolean hasStreamerUpdated() throws RemoteException;
}
```

Announcement class:

```
package model;

import java.io.Serializable;

/**
 *
 * @author BradleyW
 */
public class Announcement implements Serializable{

    private final String announcment;
    private final String announcerName;

    public Announcement(String username, String msg)
    {
        this.announcerName = username;
        this.announcment = msg;
    }

    public String getText(){
        return announcment;
    }

    public String getName() {
        return announcerName;
    }
}
```

Chatlog class:

```
package model;

import java.io.Serializable;
import java.util.LinkedList;
import java.util.concurrent.CopyOnWriteArrayList;

/**
 *
 * @author BradleyW
 */
public class Chatlog implements Serializable {

    public CopyOnWriteArrayList<Message> textLog;

    public Chatlog()
```

```
  {
    textLog = new CopyOnWriteArrayList<>();
  }

  public void addMessage(Message Message)
  {
    textLog.add(Message);
  }

  public String viewEntries()
  {
    String allEntries = "";
    int i = 0;
    for (Message msg : textLog)
    {
      allEntries = allEntries + i + ": " + msg.msgDisplay() + "\n";
      i++;
    }
    return allEntries;
  }


  public LinkedList<Message> getAllEntriesAfter(int lstMsgRecievedLine)
  {
    return new LinkedList<Message>(textLog.subList(lstMsgRecievedLine, textLog.size()));
  }

}
```

ConnectionEntry class:

```
package model;

import java.io.Serializable;

/**
 *
 * @author BradleyW
 */
public class ConnectionEntry implements Serializable{

  private String name;
  private String desc;

  public ConnectionEntry(String conName, String conDesc){
    this.name = conName;
    this.desc = conDesc;
  }

  public String getName()
  {
    return name;
  }

  public String getDesc()
  {
    return desc;
  }
```

```
}
```

ConnectionsLog class:

```
package model;

import java.io.Serializable;
import java.util.HashMap;
import java.util.LinkedList;

/**
 *
 * @author BradleyW
 */
public class ConnectionsLog implements Serializable{

   private HashMap<String, ConnectionEntry> connections;

   public ConnectionsLog()
   {
      connections = new HashMap<>();
   }

   public String getConnectionsDisplay()
   {
      String returnString = "";
      int indexBuilder = 0;
      for (ConnectionEntry connect : connections.values())
      {
         returnString = returnString +
               indexBuilder + " " + connect.getName() +
               "\n";
         indexBuilder++;
      }
      return returnString;
   }

   public ConnectionEntry getConnection(String connectionName)
   {
      return connections.get(connectionName);
   }

   public LinkedList<ConnectionEntry> getAllConnections()
   {
      return new LinkedList(connections.values());
   }

   public void removeConnection(String connectionName)
   {
      connections.remove(connectionName);
   }

   public void addConnection(String name, String desc)
   {
      ConnectionEntry newConnection = new ConnectionEntry(name, desc);
      connections.put(name, newConnection);
   }
```

```
  public ConnectionEntry getConnection(int connectionIndex)
  {
    int index = 0;
    for (ConnectionEntry conn : connections.values())
    {
      if (index==connectionIndex)
      {
        return conn;
      }
    }
    return null;
  }

  public String getConnectionName(int connectionIndex)
  {
    ConnectionEntry entry = getConnection(connectionIndex);
    return entry.getName();
  }

  public String getConnectionDetails(int connectionIndex)
  {
    ConnectionEntry entry = getConnection(connectionIndex);
    return entry.getDesc();
  }

  public int getConnectionsSize()
  {
    return connections.size();
  }

}
```

Message class:

```
package model;

import java.io.Serializable;
import java.util.Date;

/**
 *
 * @author BradleyW
 */
public class Message implements Serializable {

  private Date timestamp;
  private String sender;
  private String recipientId;
  private String msg;

  private MessageType messageType;

  public Message(String isender, String irecipientId, String imsg, int messageTypei)
  {
    timestamp = new Date();
    sender = isender;
    recipientId = irecipientId;
    msg = imsg;
```

```
        messageType = new MessageType(messageTypei);
    }

    public void setMsgText(String imsg)
    {
        this.msg = imsg;
    }

    public String msgDisplay()
    {
        String returnString = "[" + timestamp.getHours() + ":" + timestamp.getMinutes() + "] " + getType() + " - " + sender + ": " + msg;
        return returnString;
    }

    public String getSenderName()
    {
        return sender;
    }

    public String getRecipientId()
    {
        return recipientId;
    }

    public String getType()
    {
        return messageType.getTypeText();
    }
}
```

MessageType class:

```
package model;

import java.io.Serializable;

/**
 *
 * @author BradleyW
 */
public class MessageType implements Serializable{

    private int type;

    public MessageType(int itype)
    {
        this.type = itype;
    }

    public String getTypeText()
    {
        String returnString = "";
        switch(type){
            case 1:  returnString = "Room";
                     break;
            case 2:  returnString = "System";
                     break;
            case 3:  returnString = "Admin";
```

```
            break;
        case 4:  returnString = "Private";
            break;
      }
      return returnString;
   }
}
```

ServerFrontpage class:

```
package model;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author BradleyW
 */
public class ServerFrontpage implements Serializable{

   private ArrayList<Announcement> announcements;

   public ServerFrontpage() {
      this.announcements = new ArrayList();
   }

   public void addNewAnnouncment(String username, String msg) {
      Announcement sd = new Announcement(username, msg);
      announcements.add(sd);
   }

   public void setFrontpage(List<Announcement> iannouncements) {
      this.announcements = new ArrayList(iannouncements);
   }

   public List<Announcement> getFrontpage() {
      return announcements;
   }
}
```

Server Project

AccountManager Class:

```
package conclave.ConclaveHandlers;

import conclave.db.Account;
import java.net.ConnectException;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.util.List;
import javax.persistence.CacheRetrieveMode;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import javax.persistence.Query;
```

```
import org.eclipse.persistence.config.QueryHints;
import util.Encryptor;

/**
 * This class will manage Account persistence and also validate passwords
 *
 * @author BradleyW
 */
public class AccountManager {

    private SecurityHandler sm;
    private static AccountManager instance;
    EntityManagerFactory emf;

    private AccountManager() {
        emf = Persistence.createEntityManagerFactory("ConclavePU");
    }

    public static AccountManager getInstance()
    {
        if (instance==null)
        {
            instance = new AccountManager();
        }
        return instance;
    }
    /**
     * Returns an Account object based on a userid
     *
     * @param userid
     * @return Account
     * @throws ConnectException
     */
    public Account getAccount(int userid) throws ConnectException {
        EntityManager manager = emf.createEntityManager();
        manager.setProperty(QueryHints.CACHE_RETRIEVE_MODE, CacheRetrieveMode.USE);
        Query query = manager.createNamedQuery("Account.findByUserid");
        query.setParameter("userid", userid);
        List<Account> accounts = query.getResultList();
        Account potAccount = accounts.get(0);
        if (potAccount != null) {
            return potAccount;
        }
        manager.close();
        return null;
    }

    /**
     * Adds an account using a username and a password. It will hash this
     * password and store it into the persistence database
     *
     * @param username
     * @param PTpassword
     * @throws ConnectException
     */
    public void addAccount(String username, String PTpassword) throws ConnectException {
        byte[] salt = new byte[16];
        try {
            SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
            sr.nextBytes(salt);
```

```
    } catch (NoSuchAlgorithmException e) {
    }
    if (!isAUser(username)) {
        String hashedPassword = Encryptor.hashPassword(PTpassword, salt);
        Account newAccount = new Account();
        newAccount.setHashedpassword(hashedPassword);
        newAccount.setUsername(username);
        newAccount.setSalt(salt);
        EntityManager em = emf.createEntityManager();
        em.setProperty(QueryHints.CACHE_STORE_MODE, CacheRetrieveMode.BYPASS);
        EntityTransaction et = em.getTransaction();
        et.begin();
        em.persist(newAccount);
        et.commit();
        em.close();
    }
}

/**
 * Gets a user object by using a username, this is a much more general
 * search of the database and allows a user to be found without knowing the
 * userid. As username uniquness is enforced, this is a perfectly acceptable
 * way to search the database
 *
 * @param username
 * @return Account
 * @throws ConnectException
 */
/**
 *
 * @param username
 * @return
 * @throws ConnectException
 */
public Account getUserByName(String username) throws ConnectException {
    Account returnedAccount = null;
    if (isAUser(username)) {
        EntityManager em = emf.createEntityManager();
        em.setProperty(QueryHints.CACHE_RETRIEVE_MODE, CacheRetrieveMode.USE);
        Query query = em.createNamedQuery("Account.findByUsername");
        query.setParameter("username", username);
        returnedAccount = (Account) query.getSingleResult();
        em.close();
    }
    return returnedAccount;
}

/**
 * Verifies a username and password, this is used to verify login details
 * before a ServerController initiates a UserInterface initilization and
 * adds it to the RMI Registry.
 *
 * @param iusername
 * @param ptPassword
 * @return
 * @throws ConnectException
 */
public boolean verifyUser(String iusername, String ptPassword) throws ConnectException {
    boolean verified = false;
    Account returnedAccount = getUserByName(iusername);
```

```
        if (returnedAccount.getUsername().equals(iusername)) {
            String hashandSalt = returnedAccount.getHashedpassword();
            byte[] salt = returnedAccount.getSalt();
            String enteredHashedPassword = Encryptor.hashPassword(ptPassword, salt);
            if (enteredHashedPassword.equals(hashandSalt)) {
                verified = true;
                //System.out.println("User " + returnedAccount.getUsername() + " has logged in.");
            }
        }
        return verified;
    }

    /**
     * Verifies if a username exists in the database.
     *
     * @param username
     * @return
     * @throws ConnectException
     */
    public boolean isAUser(String username) throws ConnectException {
        EntityManager em = emf.createEntityManager();
        em.setProperty(QueryHints.CACHE_RETRIEVE_MODE, CacheRetrieveMode.USE);
        Query query = em.createNamedQuery("Account.findByUsername");
        query.setParameter("username", username);
        List<Account> accounts = query.getResultList();
        for (Account account : accounts) {
            String tempUsrName = account.getUsername();
            if (tempUsrName.equals(username)) {
                return true;
            }
        }
        em.close();
        return false;
    }

    /**
     * Lists all the users on a database, this method isn't used in Conclave,
     * but during development allowed testing and provides a primitive java
     * interface.
     *
     * @return
     * @throws ConnectException
     */
    public String listAllUsers() throws ConnectException {
        String allUsers = "";
        EntityManager manager = emf.createEntityManager();
        Query query = manager.createNamedQuery("Account.findAll");
        List<Account> accounts = query.getResultList();
        for (Account account : accounts) {
            allUsers = allUsers + account.toString();
        }
        manager.close();
        return allUsers;
    }
}
```

ClientHandler Class:

```
package conclave.ConclaveHandlers;
```

```
import conclave.controllers.ServerController;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.ConnectException;
import java.net.Socket;
import java.util.logging.Level;
import java.util.logging.Logger;
import util.Encryptor;

/**
 * This ClientHandler class will receive and send Socket messages, this will
 * handle request like: PING, LOGIN (username) (password), SETUP-ACCOUNT
 * (username) (password).
 *
 * The response will return a HTTP request to the socket, and using a Switch
 * method will also return a String response.
 *
 * @author BradleyW
 */
public class ClientHandler implements Runnable {

    private Socket socket;
    private ServerController server;
    private String skey;
    private int skeyUser;

    public ClientHandler(Socket sock) {
        this.socket = sock;
        this.server = ServerController.getInstance();
    }

    /**
     * This run handles the request, returns a response and then closes the
     * socket.
     */
    @Override
    public void run() {
        String response = "INVALID REQUEST";
        try {
            String request = recieveRequest();
            SecurityHandler sm = new SecurityHandler();
            if (request.length() > 5) {
                skeyUser = Integer.parseInt(request.substring(0, 5));
                if (sm.isAKey(skeyUser)) {
                    skey = sm.getKey(skeyUser);
                    request = request.substring(5, request.length());
                    String plaintextRequest = Encryptor.decrypt(request, skey);
                    System.out.println(plaintextRequest);
                    response = handleRequest(plaintextRequest);
                } else {
                    response = responseCode(403) + "Invalid key given";
                }
```

```
            Logger.getLogger(ClientHandler.class.getName()).log(Level.INFO,
                "ClientHandler: Thread '"
                + Thread.currentThread().toString()
                + "' received message: " + request
                + "' response sent: " + response);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    sendResponse(response);
}

/**
 * sends a response to an already established Socket.
 *
 * @param msg
 * @return
 */
public boolean sendResponse(String msg) {
    try {
        String response = "";
        if (skey != null)//Checks to see if we have a session key for this communication
        {
            response = Encryptor.encrypt(msg, skey);
        } else { //here the SKey was invalid, and this will usually be a invalid key msg.
            response = msg;
        }
        response = response + "\n";
        BufferedOutputStream bos = new BufferedOutputStream(socket.getOutputStream());
        OutputStreamWriter osw = new OutputStreamWriter(bos, "UTF-8");
        osw.write(response);
        osw.flush();
    } catch (IOException ex) {
        Logger.getLogger(ClientHandler.class
                .getName()).log(Level.SEVERE, null, ex);
    }
    return false;
}

/**
 * Recieves a request and returns the request as a string.
 *
 * @return
 * @throws IOException
 */
public String recieveRequest() throws IOException {
    InputStreamReader insr = new InputStreamReader(socket.getInputStream());
    BufferedReader br = new BufferedReader(insr);
    String entireRequest = "";
    String nextLine = null;
    try {
        while (true) {
            Thread.sleep(10);
            if ((nextLine = br.readLine()) != null) {
                entireRequest = entireRequest + nextLine;
                if (!br.ready()) {
```

```
                break;
            } else {
                entireRequest = entireRequest + "\n";

            }
        }
    }

    } catch (InterruptedException ex) {
        Logger.getLogger(ClientHandler.class
                .getName()).log(Level.SEVERE, null, ex);
    } catch (IOException ex) {
        Logger.getLogger(ClientHandler.class
                .getName()).log(Level.SEVERE, null, ex);
    }
    return entireRequest;
}

/**
 * deals with a string request and returns a valid response.
 *
 * @param request
 * @return
 */
private String handleRequest(String request) {
    String returnMsg = "";
    int responseCode = 502;
    if (request != null) {
        String[] commandWords = request.split("\\s+");
        if (commandWords.length <= 0) {
            responseCode = 400;
            returnMsg = "Empty Request";
        } else if (commandWords[0].equals("PING")) {
            responseCode = 100;
            returnMsg = "Server is ready for connections";
        } else if (commandWords[0].equals("LOGIN")) {
            if (commandWords.length <= 2) {
                returnMsg = "No Username/Password given";
                responseCode = 400;
            } else if (commandWords.length == 3) {
                try {
                    String username = commandWords[1];
                    String password = commandWords[2];
                    if (!server.isAUser(username)) {
                        responseCode = 404;
                        returnMsg = "A user by that username could not be found";
                    } else if (server.isUserLoggedIn(username)) {
                        responseCode = 423;
                        returnMsg = "That user is already logged in";
                    } else if (server.isBanned(username)) {
                        responseCode = 403;
                        returnMsg = "That user is banned from the server";
                    } else {
                        boolean loggedIn = server.login(username, password);
                        if (loggedIn) {
                            responseCode = 100;
```

```
                if (server.isAAdmin(username)) {
                    returnMsg = "You have logged in as an Admin: " + username;
                } else {
                    returnMsg = "You have logged in as a User: " + username;
                }
                SecurityHandler sh = new SecurityHandler();
                sh.logKeyuse(skeyUser, username);
            } else {
                responseCode = 401;
                returnMsg = "Incorrect username/password";

            }
        }
    } catch (ConnectException ex) {
        Logger.getLogger(ClientHandler.class
                .getName()).log(Level.SEVERE, null, ex);
    }
  }
} else if (commandWords[0].equals("SETUP-ACCOUNT")) {
  System.out.println("setting up account");
  if (commandWords.length <= 2) {
    returnMsg = "No username or password given";
    responseCode = 400;
  } else if (commandWords.length == 3) {
    String username = commandWords[1];
    String password = commandWords[2];
    if (username.length() > 5 && password.length() > 5) {
      if (username.length() < 15 && password.length() < 15) {
        try {
          if (server.isAUser(username)) {
            returnMsg = "That user already exists";
            responseCode = 202;
          } else {
            boolean creationSuccess = false;
            try {
              server.createAccount(username, password);
              creationSuccess = server.isAUser(username);

            } catch (ConnectException ex) {
              Logger.getLogger(ClientHandler.class
                    .getName()).log(Level.SEVERE, null, ex);
            }
            responseCode = 400;
            returnMsg = "Incorrect account creation details";
            if (creationSuccess) {
              responseCode = 100;
              returnMsg = "Account creation success";
            }
          }
        } catch (ConnectException e) {

        }
      } else {
        responseCode = 400;
        returnMsg = "Username and Password must be less than 15 characters long";
      }
```

```
            } else {
                responseCode = 400;
                returnMsg = "Username and Password must be atleast 5 characters long";
            }
          }
        } else {
          responseCode = 400;
          returnMsg = "That request is not recognized.";
        }
    }
    return responseCode(responseCode) + returnMsg;
}

/**
 * changes a response code into a String response.
 *
 * @param code
 * @return
 */
public String responseCode(int code) {
    String returnString;
    switch (code) {
        case 100:
            returnString = "100 REQUEST VALID";
            break;
        case 201:
            returnString = "201 ACCOUNT CREATION SUCCESS";
            break;
        case 202:
            returnString = "202 ACCOUNT CREATE UNSUCCESSFUL";
            break;
        case 400:
            returnString = "400 REQUEST SYNTAX ERROR";
            break;
        case 401:
            returnString = "401 INCORRECT LOGIN DETAILS";
            break;
        case 404:
            returnString = "404 CANNOT LOCATE RESOURCE";
            break;
        case 501:
            returnString = "501 NOT YET IMPLEMENTED";
            break;
        case 502:
            returnString = "502 SERVER ERROR";
            break;
        case 403:
            returnString = "403 NOT PERMITTED";
            break;
        case 423:
            returnString = "423 USER ALREADY LOGGED IN";
            break;
        default:
            returnString = "502 SERVER ERROR";
            break;
    }
```

```
        returnString = returnString + " ";
        return returnString;
    }
}
```

RoomManager Class:

```
package conclave.ConclaveHandlers;
import conclave.db.Room;
import conclave.model.ConferenceRoom;
import model.ConnectionsLog;
import conclave.model.TextRoom;
import conclaveinterfaces.IConclaveRoom;
import java.rmi.AlreadyBoundException;
import java.rmi.NoSuchObjectException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import javax.persistence.Query;
import util.Encryptor;

/**
 * This class can be likened to the AcccountManager, in that it persists objects
 * to a database, and validates room passwords using the hash method. This
 * manager is also responsible for maintaining a list of all the rooms and their
 * information, passing roomlistings to users. Users own this object as it
 * allows them to always be aware of any room connection they may be able to
 * connect to.
 *
 * @author BradleyW
 */
public class RoomManager {

    private ConnectionsLog roomConnections;
    private SecurityHandler sm;
    private HashMap<String, IConclaveRoom> hostedRooms;
    private final ArrayList<String> supportedRoomtypes = new ArrayList<>();
    private ArrayList<String> mutedUsers = new ArrayList<>();
    private static final Logger log = Logger.getLogger(RoomManager.class.getName());
    private static RoomManager instance;
    EntityManagerFactory emf;

    /**
     * Supported room types are initilized, for demonstration purposes, in the
     * constructor. New rooms that the manager could support must be added here
```

```
     * if any new rooms are developed.
     */
    private RoomManager() {
        roomConnections = new ConnectionsLog();
        sm = new SecurityHandler();
        supportedRoomtypes.add("ConferenceRoom");
        supportedRoomtypes.add("TextRoom");
        hostedRooms = new HashMap<>();
        emf = Persistence.createEntityManagerFactory("ConclavePU");
    }

    /**
     * As we want one unique instance of this object, we must employ a
     * singleton pattern.
     *
     * @return
     */
    public static RoomManager getInstance() {
        if (instance == null) {
            instance = new RoomManager();
        }
        return instance;
    }

    /**
     * This mounts an open room onto the server, that does not need a password
     * to enter. This type of room is not persisted to the database.
     *
     * @param roomname
     * @param roomType
     * @return
     * @throws RemoteException
     */
    public boolean mountOpenRoom(String roomname, int roomType) throws RemoteException {
        boolean ok = false;
        try {
            IConclaveRoom room = null;
            switch (String.valueOf(roomType)) {
                case "1":
                    room = new TextRoom(roomname);
                    break;
                case "2":
                    room = new ConferenceRoom(roomname);
                    break;
            }
            if (room != null && !isARoom(roomname)) {
                room.openRoom();
                hostedRooms.put(roomname, room);
                roomConnections.addConnection(roomname, room.getInfo());
                Registry reg = LocateRegistry.getRegistry(9807);
                reg.bind(room.getRoomName(), room);
                log.log(Level.INFO, "An open room {0} has been added to the registry", roomname);
                ok = true;
            }
        } catch (AlreadyBoundException e) {
```

```
    }
    return ok;
  }

  /**
   * Creates a room and persists it to the database, this room is not added to
   * the registry.
   *
   * @param roomname
   * @param password
   * @param roomType
   * @return
   * @throws RemoteException
   */
  public boolean createRoom(String roomname, String password, int roomType) throws RemoteException {
    boolean success = false;
    Room newPersistenceRoom = new Room();
    byte[] salt = new byte[16];
    try {
      SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
      sr.nextBytes(salt);
    } catch (NoSuchAlgorithmException e) {
    }
    try {
      if (!isARoom(roomname)) {
        String hashedPassword = Encryptor.hashPassword(password, salt);
        newPersistenceRoom.setHashedpassword(hashedPassword);
        newPersistenceRoom.setRoomname(roomname);
        newPersistenceRoom.setSalt(salt);
        newPersistenceRoom.setRoomtype(String.valueOf(roomType));
        EntityManager em = emf.createEntityManager();
        EntityTransaction et = em.getTransaction();
        et.begin();
        em.persist(newPersistenceRoom);
        et.commit();
        em.close();
        log.log(Level.INFO, "A new room {0} has been persisted to the DB", roomname);
        success = true;
      }
    } catch (RemoteException e) {
    }
    return success;
  }

  /**
   * This will load a room into the registry, (By roomname) from the
   * persistence database, and allow it to be accessed by users.
   *
   * @param roomname
   * @return
   * @throws RemoteException
   */
  public boolean loadRoom(String roomname) throws RemoteException {
    boolean success = false;
    try {
      if (isARoom(roomname)) {
```

```
            IConclaveRoom room = getConclaveRoom(roomname);
            Registry reg = LocateRegistry.getRegistry(9807);
            reg.bind(room.getRoomName(), room);
            room.openRoom();
            roomConnections.addConnection(roomname, room.getInfo());
            hostedRooms.put(roomname, room);
            success = true;
            log.log(Level.INFO, "A persisted room: {0} has been loaded to the registry", roomname);
        }
    } catch (RemoteException e) {
        e.printStackTrace();
    } catch (AlreadyBoundException e) {
        e.printStackTrace();
    }
    return success;
}

/**
 * this will list all the rooms on the registry in the form of a returned String.
 * This is useful for bug fixing.
 * @return
 */
public String listAllRooms() {
    String returnString = "";
    try {
        Registry reg = LocateRegistry.getRegistry(9807);
        String[] rooms = reg.list();
        for (String room : rooms) {
            returnString+= room;
        }
    } catch (RemoteException e) {

    }
    return returnString;
}

/**This will return a Connectionlog of all active rooms, this enables users to
 * get an up-to date log of all rooms when they join the server, or leave a room.
 * Updates to already connected users are done elsewhere.
 *
 * @return
 * @throws RemoteException
 */
public ConnectionsLog returnRooms() throws RemoteException {
    return roomConnections;
}

/**
 * Returns a room based on a roomname, from the persistence database.
 * It will also assign a specific room implementation to the object it passes.
 * When implementing a new room, care must be made and the new type added to
 * the switch.
 *
 * @param roomname
 * @return
 * @throws RemoteException
```

```
 */
public IConclaveRoom getConclaveRoom(String roomname) throws RemoteException {
   IConclaveRoom returnRoom = null;
   if (hostedRooms.get(roomname) != null) {
      returnRoom = hostedRooms.get(roomname);
   } else {
      EntityManager manager = emf.createEntityManager();
      Query query = manager.createNamedQuery("Room.findAll");
      List<Room> rooms = query.getResultList();
      for (Room room : rooms) {
         String tmpRoomName = room.getRoomname();
         if (tmpRoomName.equals(roomname)) {
            String roomType = room.getRoomtype();
            switch (roomType) {
               case "1":
                  returnRoom = new TextRoom(roomname);
                  break;
               case "2":
                  returnRoom = new ConferenceRoom(roomname);
                  break;
            }
         }
      }
   }
   return returnRoom;
}

/**
 * Returns a boolean based on if the username is in the RoomManager muted users list
 * @param username
 * @return
 * @throws RemoteException
 */
public boolean isMuted(String username) throws RemoteException {
   boolean mutedStatus = false;
   if (mutedUsers.contains(username)) {
      mutedStatus = true;
   }
   return mutedStatus;
}

/** Kicks a user from a room. The banned flag is depreciated, but
 * may still be implemented.
 *
 * @param username
 * @param banned
 * @throws RemoteException
 */
public void kickUser(String username, boolean banned) throws RemoteException {
   for (IConclaveRoom croom : hostedRooms.values()) {
      if (croom.hasUser(username)) {
         croom.kickUser(username);
      }
   }
}
```

```
/**Unmutes a user by a username from the mutedUser array.
 *
 * @param username
 * @throws RemoteException
 */
public void uncensorUser(String username) throws RemoteException {
   if (mutedUsers.contains(username))
   {
      mutedUsers.remove(username);
      for (IConclaveRoom croom : hostedRooms.values()) {
         croom.uncensorUser(username);
      }
   }
}

/**Mutes a user by a username
 *
 * @param username
 * @throws RemoteException
 */
public void censorUser(String username) throws RemoteException {
   mutedUsers.add(username);
   for (IConclaveRoom croom : hostedRooms.values()) {
      croom.addCensoredUser(username);
   }
}

/**
 * If the room exists, returns true.
 *
 * @param roomname
 * @return
 * @throws RemoteException
 */
public boolean isARoom(String roomname) throws RemoteException {
   EntityManager manager = emf.createEntityManager();
   Query query = manager.createNamedQuery("Room.findAll");
   List<Room> rooms = query.getResultList();
   for (Room room : rooms) {
      String tmpRoomName = room.getRoomname();
      if (tmpRoomName.equals(roomname)) {
         return true;
      }
   }
   return false;
}

/**
 * Removes a room from the Connections Log.
 *
 * @param roomname
 * @return
 * @throws RemoteException
 */
public boolean deleteRoom(String roomname) throws RemoteException {
   boolean successful = false;
```

```
    //JPA code
      roomConnections.removeConnection(roomname);
    return successful;
  }

  /**
   * Stops a room, and removes it from the connections log aswell as from the
   * registry.
   *
   * @param roomname
   * @throws RemoteException
   */
  public void stopRoom(String roomname) throws RemoteException {
    if (isARoom(roomname) && hostedRooms.containsKey(roomname)) {
      IConclaveRoom room = getConclaveRoom(roomname);
      room.stopRoom();
      roomConnections.removeConnection(roomname);
      hostedRooms.remove(roomname);
      log.log(Level.INFO, "Room: {0} has been stopped", roomname);
    }
  }

  /**
   * Returns true if the specified roomname has a password.
   * @param roomname
   * @return
   * @throws RemoteException
   */
  public boolean hasPassword(String roomname) throws RemoteException {
    boolean has = false;
    if (isARoom(roomname)) {
      Room room = getRoom(roomname);
      if (room.getHashedpassword() != null) {
        has = true;
      }
    }
    return has;
  }

  /**
   * Validates a room bassed on a roomname and a password. This password
   * is hashed and salted against the roomEntry password. Returns true
   * if a correct password, and false for bad password.
   *
   * @param roomname
   * @param password
   * @return
   * @throws RemoteException
   */
  public boolean valdiateRoom(String roomname, String password) throws RemoteException {
    boolean validated = false;
    if (isARoom(roomname) && hasPassword(roomname)) {
      Room room = getRoom(roomname);
      byte[] salt = room.getSalt();
      String enteredHashedPassword = Encryptor.hashPassword(password, salt);
      if (room.getHashedpassword().equals(enteredHashedPassword)) {
```

```
            validated = true;
        } else {
        }
    }
    return validated;
}

/**
 * Gets a room from the persistence database based on a roomname.
 *
 * @param roomname
 * @return
 * @throws RemoteException
 */
public Room getRoom(String roomname) throws RemoteException {
    Room returnedRoom = null;
    EntityManager manager = emf.createEntityManager();
    Query query = manager.createNamedQuery("Room.findByRoomname");
    query.setParameter("roomname", roomname);
    List<Room> rooms = query.getResultList();
    for (Room room : rooms) {
        String tmpRoomName = room.getRoomname();
        if (tmpRoomName.equals(roomname)) {
            returnedRoom = room;
        }

    }
    manager.close();
    return returnedRoom;
}

/**
 * Returns a collection of all the roomnames.
 * @return
 */
public List<String> getAllRoomNames()
{
    EntityManager manager = emf.createEntityManager();
    Query query = manager.createNamedQuery("Room.findAll");
    List<Room> rooms = query.getResultList();
    List<String> returnCollection = new ArrayList();
    for (Room room : rooms) {
        returnCollection.add(room.getRoomname());
    }
    return returnCollection;
}

/**
 * Sets a room open flag to true, it will also update the connections log.
 *
 * @param roomname
 * @throws RemoteException
 */
public void openRoom(String roomname) throws RemoteException {
    IConclaveRoom room = hostedRooms.get(roomname);
    room.openRoom();
```

```
    roomConnections.removeConnection(roomname);
    roomConnections.addConnection(roomname, room.getInfo());
    log.log(Level.INFO, "Room: {0} has been opened", roomname);
  }

  /**
   * Closes a room based on a roomname. This room will not let any new users connect to it, and is used to
   * prevent interruptions to a meeting.
   *
   * @param roomname
   * @throws RemoteException
   */
  public void closeRoom(String roomname) throws RemoteException {
    IConclaveRoom room = hostedRooms.get(roomname);
    room.closeRoom();
    roomConnections.removeConnection(roomname);
    roomConnections.addConnection(roomname, room.getInfo());
    log.log(Level.INFO, "Room: {0} has been closed", roomname);
  }

  /**
   * Returns a collection of all the roomnames, this is used to generally
   * find out what rooms are available to try and join.
   *
   * @return
   * @throws RemoteException
   */
  public List<String> getAllLoadedRoomnames() throws RemoteException {
    return new ArrayList(hostedRooms.keySet());
  }


  /**
   * Finds out all the room types that this manager is capable of supporting.
   * @return
   * @throws RemoteException
   */
  public List<String> getAllSupportedRoomTypes() throws RemoteException {
    return supportedRoomtypes;
  }

  public int roomsAmount(){
    return hostedRooms.size();
  }

  public void stopRooms(){
    for (IConclaveRoom room : hostedRooms.values())
    {
      try {
        UnicastRemoteObject.unexportObject(room, true);
      } catch (NoSuchObjectException ex) {
        Logger.getLogger(RoomManager.class.getName()).log(Level.SEVERE, null, ex);
      }
    }
  }
```

```
}
```

SecurityHandler Class:

```
package conclave.ConclaveHandlers;

import conclave.db.Sessionkeys;
import conclave.db.Userkeys;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.util.ArrayList;
import java.util.List;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.NoResultException;
import javax.persistence.Persistence;
import javax.persistence.Query;

/**
 * This class is responsible for hashing passwords and encrypting Strings using
 * DES Encryption.
 *
 * @author BradleyW
 */
public class SecurityHandler {

  EntityManagerFactory emf;

  /**
   * Initiates a AES key to initiate session keys.
   */
  public SecurityHandler() {
    emf = Persistence.createEntityManagerFactory("ConclavePU");
  }

  public void logKeyuse(int keyid, String username) {
    EntityManager em = emf.createEntityManager();
    Query query = em.createNamedQuery("Userkeys.findByKeyId");
    query.setParameter("keyid", getKeyObj(keyid));
    boolean userAlreadyLogged = false;
    List<Userkeys> users = query.getResultList();
    for (Userkeys ukers : users) {
      if (ukers.getUsername().equals(username)) {
        userAlreadyLogged = true;
      }
    }
    if (!userAlreadyLogged) {
      Userkeys uk = new Userkeys();
      Sessionkeys key = getKeyObj(keyid);
      uk.setKeyid(key);
      uk.setUsername(username);
      EntityTransaction et = em.getTransaction();
      et.begin();
      em.persist(uk);
      et.commit();
```

```
    }
  }

  public Sessionkeys getKeyObj(int keyid) {
    Sessionkeys sk = null;
    try {
      EntityManager em = emf.createEntityManager();
      Query query = em.createNamedQuery("Sessionkeys.findByKeyidentify");
      query.setParameter("keyidentify", keyid);
      sk = (Sessionkeys) query.getSingleResult();
    } catch (NoResultException e) {

    }
    return sk;
  }

  public String getKey(int keyid) {
    String returnString = "";
    try {
      EntityManager em = emf.createEntityManager();
      Query query = em.createNamedQuery("Sessionkeys.findByKeyidentify");
      query.setParameter("keyidentify", keyid);
      Sessionkeys sk = (Sessionkeys) query.getSingleResult();
      returnString = sk.getKeystring();
    } catch (NoResultException e) {

    }
    return returnString;
  }

  public int generateSecretKey() throws NoSuchAlgorithmException {
    final SecureRandom prng = new SecureRandom();
    final byte[] aes128KeyData = new byte[128 / Byte.SIZE];
    prng.nextBytes(aes128KeyData);
    final SecretKey aesKey = new SecretKeySpec(aes128KeyData, "AES");
    String stringkey = encodeHex(aesKey.getEncoded());
    Sessionkeys sk = new Sessionkeys();
    sk.setKeystring(stringkey);
    EntityManager em = emf.createEntityManager();
    EntityTransaction et = em.getTransaction();
    et.begin();
    em.persist(sk);
    et.commit();
    em.close();
    return sk.getKeyidentify();
  }

  public boolean isAKey(int keyId) {
    Boolean ok = false;
    EntityManager em = emf.createEntityManager();
    Query query = em.createNamedQuery("Sessionkeys.findByKeyidentify");
    query.setParameter("keyidentify", keyId);
    List<Sessionkeys> sks = query.getResultList();
    for (Sessionkeys sk : sks) {
      if (sk.getKeyidentify().equals(keyId));
      {
```

```
            ok = true;
        }
    }
    return ok;
}

public String encodeHex(byte[] encode) {
    String hexValue = "";
    char[] hexChars = new char[encode.length * 2];
    char[] hexArray = "0123456789ABCDEF".toCharArray();
    for (int j = 0; j < encode.length; j++) {
        int v = encode[j] & 0xFF;
        hexChars[j * 2] = hexArray[v >>> 4];
        hexChars[j * 2 + 1] = hexArray[v & 0x0F];
    }
    hexValue = new String(hexChars);
    return hexValue;
}

public byte[] decodeHex(String hexValue) {
    {
        String s = (String) hexValue;
        int len = s.length();
        byte[] data = new byte[len / 2];
        for (int i = 0; i < len; i += 2) {
            data[i / 2] = (byte) ((Character.digit(s.charAt(i), 16) << 4)
                    + Character.digit(s.charAt(i + 1), 16));
        }
        return data;
    }
}

public List<String> getAllKeys() {
    EntityManager em = emf.createEntityManager();
    Query query = em.createNamedQuery("Sessionkeys.findAll");
    List<Sessionkeys> sks = query.getResultList();
    List<String> keyValues = new ArrayList();
    for (Sessionkeys sesskeys : sks) {
        keyValues.add(sesskeys.getKeystring() + String.format("%05d", sesskeys.getKeyidentify()));
    }
    return keyValues;

}

public void revokeKey(String stringKey) {
    try {
        EntityManager em = emf.createEntityManager();
        Query query = em.createNamedQuery("Sessionkeys.findByKeystring");
        query.setParameter("keystring", stringKey);
        Sessionkeys sk = (Sessionkeys) query.getSingleResult();
        EntityTransaction et = em.getTransaction();
        et.begin();
        em.remove(sk);
        et.commit();
    } catch (NoResultException e) {
```

```
    }
  }

  public List<String> getKeyUsers(String stringkey) {
    EntityManager em = emf.createEntityManager();
    Query findKeyID = em.createNamedQuery("Sessionkeys.findByKeystring");
    findKeyID.setParameter("keystring", stringkey);
    Sessionkeys sk = (Sessionkeys) findKeyID.getSingleResult();
    Query findUsers = em.createNamedQuery("Userkeys.findByKeyId");
    findUsers.setParameter("keyid", sk);
    List<Userkeys> users = findUsers.getResultList();
    List<String> usernames = new ArrayList();
    for (Userkeys usrk : users) {
      usernames.add(usrk.getUsername());
    }
    return usernames;
  }
}
```

AdminInterfaceImpl Class:

```
package conclave.controllers;

import conclave.ConclaveHandlers.RoomManager;
import conclave.db.Account;
import conclaveinterfaces.IAdminInterface;
import java.rmi.RemoteException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import model.Message;

/**AdminInterface implementation classe, this implements needed Admin functionality.
 * As the admin interface is also responsible for normal user interactions, we
 * extend the UserInterfaceImpl object which implements these interactions.
 *
 * @author BradleyW
 */
public class AdminInterfaceImpl extends UserInterfaceImpl implements IAdminInterface{

  //AdminInterface uses a RoomManager to modify rooms.
  private final RoomManager roomManager;
  //AdminInterface uses a ServerController objec to control Conclave.
  private final ServerController serverController;
  private Logger log = Logger.getLogger(AdminInterfaceImpl.class.getName());

  /**
   * Initiating a AdminInterfaceImpl requires a usual Account object, and the ServerController instance.
   * @param account
   * @param instance
   * @throws RemoteException
   */
  public AdminInterfaceImpl(Account account, ServerController instance) throws RemoteException {
    super(account);
    roomManager = RoomManager.getInstance();
    serverController = instance;
  }
```

```
/**
 * Adds an open room to the registry. It takes in a roomname and a roomtype,
 * the RoomManager is responsible for creating the right Object
 * @param roomname
 * @param roomType
 * @throws RemoteException
 */
@Override
public void addRoom(String roomname, int roomType) throws RemoteException{
   try {
      roomManager.mountOpenRoom(roomname, roomType);
      serverController.updateAllClientsConnections(); //Update all clients connections of the new state change.
      log.log(Level.FINE, "Admin: {0} has mounted a new open room: {1}", new Object[] {this.getUsername(), roomname});
   } catch (RemoteException ex) {
     log.log(Level.SEVERE, null, ex);
   }
}

/**
 * Adds a room with a password to the Database and registry, it uses the same parameters
 * as the method it overloads, except a password.
 * @param roomname
 * @param roomType
 * @param roompassword
 * @throws RemoteException
 */
@Override
public void addRoom(String roomname, int roomType, String roompassword) throws RemoteException{
   try {
      roomManager.createRoom(roomname, roompassword, roomType);
      roomManager.loadRoom(roomname);
      serverController.updateAllClientsConnections(); //Update all clients connections of the new state change.
      log.log(Level.FINE, "Admin: {0} has persisted a new room: {1}", new Object[] {this.getUsername(), roomname});
   } catch (RemoteException ex) {
      log.log(Level.SEVERE, null, ex);
   }
}

/**
 * Sends an Admin message to the username, it users the Message class as this is more interactive.
 * @param msg
 * @param username
 * @throws RemoteException
 */
@Override
public void sendAdminMessage(Message msg, String username) throws RemoteException {
   serverController.alertUser(msg, username);
   log.log(Level.FINE, "Admin: {0} has sent a admin message [{1}]: to {2}.", new Object[] {this.getUsername(), msg, username});
}

/**
 * Removes a room from the registry.
 *
 * @param roomname
 * @throws RemoteException
 */
```

```
@Override
public void removeRoom(String roomname) throws RemoteException{
   try {
      roomManager.deleteRoom(roomname);
      serverController.updateAllClientsConnections(); //Update all clients connections of the new state change.
      log.log(Level.FINE, "Admin: {0} has removed room: {1}",new Object[] {this.getUsername(), roomname});
   } catch (RemoteException ex) {
      log.log(Level.SEVERE, null, ex);
   }
}

/**
 * Kicks a user from the server, and if the banned flag is set, it bans them from ever
 * making a connection under that username
 *
 * @param username
 * @param banned
 * @throws RemoteException
 */
@Override
public void kickUser(String username, boolean banned) throws RemoteException{
   try {
      roomManager.kickUser(username, banned);
      if (banned)
      {
          serverController.banUser(username);
      }
      serverController.updateAllClientsConnections(); //Updates all clients connections
      log.log(Level.FINE, "Admin: {0} has kicked the user: {1}. Banned? ({3})",new Object[] {this.getUsername(), username, banned});
   } catch (RemoteException ex) {
      log.log(Level.SEVERE, null, ex);
   }
}

/**
 * Closes a room to new connections.
 *
 * @param roomname
 * @throws RemoteException
 */
@Override
public void closeRoom(String roomname) throws RemoteException{
   try {
      roomManager.closeRoom(roomname);
      serverController.updateAllClientsConnections();//Update all clients connections of the new state change.
      log.log(Level.FINE, "Admin: {0}, has close the room: {1}", new Object[] {this.getUsername(), roomname});
   } catch (RemoteException e)
   {
      log.log(Level.SEVERE, null, e);
   }
}

/**
 * Closes a room to new connections.
 *
 * @param roomname
```

```
 * @throws RemoteException
 */
@Override
public void openRoom(String roomname) throws RemoteException{
   try {
      roomManager.openRoom(roomname);
      serverController.updateAllClientsConnections();//Update all clients connections of the new state change.
      log.log(Level.FINE, "Admin: {0}, has opened the room: {1}", new Object[] {this.getUsername(), roomname});
   } catch (RemoteException ex) {
      log.log(Level.SEVERE, null, ex);
   }
}

/**
 * Unmutes a user, allowing them to chat and stream.
 *
 * @param username
 * @throws RemoteException
 */
@Override
public void uncensorUser(String username) throws RemoteException {
   roomManager.uncensorUser(username);
   serverController.alertUser(new Message(getUsername(), username, "You have been unmuted", 3), username);
}

/**
 * Mutes a user, preventing them from chatting and streaming.
 * @param username
 * @throws RemoteException
 */
@Override
public void censorUser(String username) throws RemoteException{
   roomManager.censorUser(username);
   serverController.alertUser(new Message(getUsername(), username, "You have been muted", 3), username);
   log.log(Level.FINE, "Admin: {0}, has censored the user: {1}", new Object[] {this.getUsername(), username});
}

/**
 * Returns the type of the User object, this is used by the client interaction object.
 * UserInterface returns 1, and Admin returns 2.
 * @return
 * @throws RemoteException
 */
@Override
public int getType() throws RemoteException
{
   return 2;
}

/**
 * Returns a collection of all roomnames.
 * @return
 * @throws RemoteException
 */
@Override
public List<String> getRoomNames() throws RemoteException {
```

```
    return roomManager.getAllLoadedRoomnames();
  }

  /**
   * Gets all supported room types, this is used by the Admin Interface view
   * to determine what rooms the admin can create.
   * @return
   * @throws RemoteException
   */
  @Override
  public List<String> getSupportedRoomTypes() throws RemoteException {
    return roomManager.getAllSupportedRoomTypes();
  }

  /**
   * Returns a collection of all connected usernames to Conclave, used by the
   * Admin Interface view to determine connections they can control.
   * @return
   * @throws RemoteException
   */
  @Override
  public List<String> getAllConnectedUsernames() throws RemoteException {
    return serverController.getAllConnectedUsernames() ;
  }

  /**
   * Posts an announcement to the Frontpage, this is only a String; but in the
   * future could use an Announcment object to allow Files to be downloaded.
   * @param msg
   * @throws RemoteException
   */
  @Override
  public void postAnnouncment(String msg) throws RemoteException {
    serverController.updateFrontpage(getUsername(), msg);
  }

  /**
   * returns if the user is muted, this is used to determine if a controller should unmute/mute.
   * @param username
   * @return
   * @throws RemoteException
   */
  @Override
  public boolean isMuted(String username) throws RemoteException {
    boolean mutedStatus = roomManager.isMuted(username);
    return mutedStatus;
  }
}
```

ServerController Class:

```
package conclave.controllers;

import util.PerformanceLogger;

import conclave.ConclaveHandlers.AccountManager;
import conclave.ConclaveHandlers.ClientHandler;
import conclave.ConclaveHandlers.RoomManager;
```

```
import conclave.ConclaveHandlers.SecurityHandler;
import conclave.db.Account;
import conclave.rmiPolicy.RMISecurityPolicyLoader;
import conclave.rmiPolicy.RegistryLoader;
import conclaveinterfaces.IUserInterface;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;

import java.io.IOException;
import java.net.BindException;
import java.net.InetAddress;
import java.net.ConnectException;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.UnknownHostException;
import java.rmi.NoSuchObjectException;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.security.NoSuchAlgorithmException;

import java.util.ArrayList;
import java.util.List;
import java.util.Timer;
import java.util.TimerTask;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.CopyOnWriteArrayList;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.logging.Level;
import java.util.logging.Logger;

import model.ConnectionsLog;
import model.Message;
import model.ServerFrontpage;

/**
 * Servercontroller, responsible for hosting the server and maintaining
 * connections, responsbile for holding Admins, banned users and the server
 * frontpage. This class is responsible for creating UserInterfaces and
 * AdminInterfaces as well as adding to the registry
 *
 * @author BradleyW
 */
public class ServerController implements Remote {

   private String name;
   private InetAddress ip;
   private int port;
   private boolean open;
   private static int serverTicks;
   private static int clientKeysGenerated;
```

```
//For performance logging, we will be exporting the performance information into a csv file.
PerformanceLogger performance;

// TO avoid Concurrency exceptions like ConcurrentModificationException, we use
//special java.util.concurrent library collections, like a conccurentHashMap, and a
//"CopyOnWriteArraylist" which creates a copy of the collection when it edits.
private final ConcurrentHashMap<String, IUserInterface> connections = new ConcurrentHashMap<>();
private final CopyOnWriteArrayList<String> serverAdmins = new CopyOnWriteArrayList();
private final CopyOnWriteArrayList<String> bannedUsers = new CopyOnWriteArrayList();

//sServer Frontpage
ServerFrontpage frontpage;
//ServerManager, handles new socket connections
ServerManager serverManager;
//AccountManager which manages persisted users.
AccountManager accountManager;
//RoomManage which handles the Room registry, and save/loads to the database
RoomManager roomManager;

private static ServerController instance;
private static final Logger log = Logger.getLogger(ServerController.class.getName());
//Socket requests per tick
private int requestsPerTicks;
//Executor handlers limit.
private static final int HANDLERS = 10;
//ExecutorService pool.
private ExecutorService pool = null;

//As we only want one server active on a JVM, we use a singleton pattern.
public static synchronized ServerController getInstance() {
   if (instance == null) {
      instance = new ServerController();
   }
   return instance;
}

private ServerController() {
   pool = Executors.newFixedThreadPool(HANDLERS);
   serverTicks = 0; //1 tick is 1 second of runtime.
   name = "LAN Conclave Server"; //hardcoded name

   //Initiate singleton patterns for the managers
   accountManager = AccountManager.getInstance();
   roomManager = RoomManager.getInstance();
   frontpage = new ServerFrontpage();
   instance = this;
   frontpage.addNewAnnouncment(name, "Welcome to " + name); //hardcoded server welcome announcement
   open = false;

   // By adding this shutdown hook we can ensure that the server exits gracefully.
   Runtime.getRuntime().addShutdownHook(new Thread() {
      @Override
      public void run() {
         stopServer();
      }
```

```
    });
}

/**
 * Disconnects a user from the server, used when banning a user.
 *
 * @param name
 */
public void disconnectUser(String name) {
    if (connections.containsKey(name)) {
        connections.remove(name);
        try {
            roomManager.kickUser(name, false);
            //register.unbind(name);
        } catch (RemoteException ex) {
            Logger.getLogger(ServerController.class.getName()).log(Level.SEVERE, null, ex);
            //} catch (NotBoundException ex)
            //{
            //   Logger.getLogger(ServerController.class.getName()).log(Level.SEVERE, null, ex);
        }
        log.log(Level.INFO, "User: {0} has terminated their connection.", name);
    }
}

/**
 * Loads the RMI policy and security policy.
 */
public void loadRMI() {
    try {
        if (System.getSecurityManager() == null) {
            RMISecurityPolicyLoader.LoadPolicy("RMISecurity.policy");
        }
        RegistryLoader.Load();
    } catch (RemoteException e) {
        log.log(Level.SEVERE, "Failed to load the RMI registry", e);
        stopServer();
    } catch (UnknownHostException ex) {
        Logger.getLogger(ServerController.class.getName()).log(Level.SEVERE, null, ex);
    }
}

/**
 * Sets a server name.
 *
 * @param iname
 */
public void setName(String iname) {
    this.name = iname;
}

/**
 * Sets a server port which Conclave runs off.
 *
 * @param port
 */
public void setPort(int port) {
```

```
    this.port = port;
}

/**
 * Sets the IP, this is sort of redundant, as the server always runs of its
 * own localhost ip.
 *
 * @param ip
 */
public void setIp(InetAddress ip) {
    this.ip = ip;
}

/**
 * Checks to see if the user is banned, use
 *
 * @param username
 * @return
 */
public boolean isBanned(String username) {
    boolean is = false;
    if (bannedUsers.contains(username)) {
        is = true;
    }
    return is;
}

public List<String> getAllKeys() {
    SecurityHandler sh = new SecurityHandler();
    List<String> keys = sh.getAllKeys();
    return keys;
}

public void revokeKey(String keyValue) {
    SecurityHandler sh = new SecurityHandler();
    sh.revokeKey(keyValue);
}

public List<String> getKeyUsers(String key) {
    SecurityHandler sh = new SecurityHandler();
    List<String> keyUsers = sh.getKeyUsers(key);
    return keyUsers;
}

/**
 * This server manager class is responsible for accepting socket connections
 * and assigning a new handler to the connection, it is started by the
 * startServer() method and closed by closeServer();
 */
public class ServerManager implements Runnable {

    private final InetAddress ip;
    private final int port;
    private boolean open;

    public ServerManager(InetAddress ip, int port) {
```

```
        this.ip = ip;
        this.port = port;
        open = true;
    }

    public boolean isOpen() {
        return open;
    }

    @Override
    public void run() {
        try {
            ServerSocket servSock = new ServerSocket(port, 50, ip); //conventional serversocket
            servSock.setReuseAddress(true);
            open = true;
            while (open) {
                Socket clientSocket = servSock.accept();
                requestsPerTicks++;//increase the requestsPerTick, for the performance analysis.
                ClientHandler handler = new ClientHandler(clientSocket);
                pool.execute(handler);//execute the new handlers runnable in the pool.
            }
        } catch (BindException e) {
            Logger.getLogger(ClientHandler.class.getName()).log(Level.INFO, "A server has failed to bind to the IP");
        } catch (IOException e) {
            Logger.getLogger(ClientHandler.class.getName()).log(Level.INFO, "A server has failed to read from a socket connection.");
        } finally {
            if (!pool.isTerminated()) {
                System.err.println("cancel non-finished tasks");
            }
            pool.shutdownNow();
        }
    }

    /**
     * Stops the server by breaking the ServerSocket accept() loop, allowing
     * the finally clause to execute.
     */
    public void stopServer() {
        open = false;
        log.log(Level.INFO, "The server has been stopped");
        pool.shutdown();
    }

} //End of ServerManager class.

//Server Controller
  /**
   * This method will start the Server, load the RMI registry and also load
   * any rooms found in the Persistence database. It will also run a
   * ConnectionManager thread which will monitor connection states and remove
   * disconnected users.
   */
  public void startServer() {
      loadRMI();
      startRooms();
      try {
```

```
            ip = InetAddress.getLocalHost(); //gets the localhost ip.
        } catch (UnknownHostException ex) {
            log.log(Level.SEVERE, "Failed to startup the server on localhost", ex);
        }
        port = 20003; //default Conclave port is 20003
        if (ip != null) {
            log.log(Level.INFO, "A LAN has been initilized.");
        }

        performance = new PerformanceLogger(name + "-" + System.currentTimeMillis(), ip, port); // new performance writer, saved to a
unique filename
        /**
         * This is responsible for logging the performance details to the csv,
         * this allows each row in the csv file to be exactly 1 second apart.
         */
        class PerformanceLogging extends TimerTask {

            public void run() {
                serverTicks++;
                performance.systemLog(serverTicks, requestsPerTicks, connections.size(), roomManager.roomsAmount());
                requestsPerTicks = 0;
            }
        }
        Timer performanceAnalysis = new Timer();
        serverManager = new ServerManager(ip, port);
        open = true; //sets the open flag to true, allowing the newly created ServerManager to run, when it is submitted.
        //This code will take the server out of the pool, allowing all pool threads to only be used by
        //Thread newThread = new Thread(new Runnable() {
        //    public void run()
        //    {
        //        serverManager.run();
        //    }
        //});
        //newThread.start();
        pool.submit(serverManager); //submits the ServerManager into the ExecutrPool
        performanceAnalysis.scheduleAtFixedRate(new PerformanceLogging(), 0, 1000);//schedule for once every second, starting immediantly.
        if (serverManager.isOpen()) { //if the server managed to startup.
            log.log(Level.INFO, "Server: {0} has been started at: {1} on port {2}, and is now open to new connections", new Object[]{name, ip,
port});
            Thread ConnectionManager = new Thread(new Runnable() { //this thread will cycle through connections and remove ones with the
Disconnected flag.
                @Override
                public void run() {
                    while (open) { //like the ServerManager, runs while server is open.
                        for (String name : connections.keySet()) {
                            try {
                                IUserInterface uiTemp = connections.get(name);
                                if (!uiTemp.isConnected()) {
                                    disconnectUser(name);
                                }
                            } catch (RemoteException e) {
                                disconnectUser(name);
                            }
                        }
                        try {
                            Thread.sleep(1000); //Server tick, in the future this should be more natural and consistent.
```

```
                } catch (InterruptedException ex) {
                    Logger.getLogger(ServerController.class
                            .getName()).log(Level.SEVERE, null, ex);
                }
            }
        }

        });
        ConnectionManager.start();
    } else {
        log.log(Level.SEVERE, "Server failed to start");
        open = false;
    }
}

/**
 * Returns a String of all connections, used as a testing method
 *
 * @return
 * @throws RemoteException
 */
public String viewAllConnections() throws RemoteException {
    String returnString = "";
    int i = 0;
    for (IUserInterface connection : connections.values()) {
        returnString = returnString + " User " + i + ":" + connection.getUsername();
        i++;
    }
    return returnString;
}

/**
 * Connects the user to a server, if the server is open and the username is
 * not banned.
 *
 * @param username
 * @return
 */
public boolean connect(String username) {
    boolean accepted = false;
    if (open && !bannedUsers.contains(username)) {
        accepted = true;
    }
    return accepted;
}

/**
 * Finds out if the username exists.
 *
 * @param name
 * @return
 * @throws java.net.ConnectException
 */
public boolean isAUser(String name) throws java.net.ConnectException {
    return accountManager.isAUser(name);
```

```
}

/**
 * If a username is an Admin.
 *
 * @param username
 * @return
 */
public boolean isAAdmin(String username) {
   boolean isAdmin = false;
   if (serverAdmins.contains(username)) {
      isAdmin = true;
   }
   return isAdmin;
}

/**
 * Updates the frontpage with a msg, the username is used for clarity.
 *
 * @param username
 * @param msg
 */
public void updateFrontpage(String username, String msg) {
   frontpage.addNewAnnouncment(username, msg);
   updateAllClientsFrontpage(username, msg);
   log.log(Level.INFO, "Admin: {0} has posted a new announcement: {1}", new Object[]{username, msg});
}

/**
 * Updates all the users frontpage, this enables controlles to push
 * frontpage announcements in realtime.
 *
 * @param username
 * @param msg
 */
public void updateAllClientsFrontpage(String username, String msg) {
   for (IUserInterface ui : connections.values()) {
      try {
         ui.updateFrontpage(username, msg);
      } catch (RemoteException ex) {
         log.log(Level.SEVERE, null, ex);
      }
   }
}

/**
 * When aaconnection changes, all clients connections log is updated from
 * this method.
 */
public void updateAllClientsConnections() {
   for (IUserInterface ui : connections.values()) {
      try {
         ui.updateConnections(roomManager.returnRooms());

      } catch (RemoteException ex) {
         Logger.getLogger(ServerController.class
```

```
                .getName()).log(Level.SEVERE, null, ex);
        }
    }
}

/**
 * Returns the current frontpage
 *
 * @return
 */
public List<String> getFrontpage() {
    return new ArrayList(frontpage.getFrontpage());
}

/**
 * Starts all rooms found in the persistence DB.
 */
public void startRooms() {
    try {
        roomManager.mountOpenRoom("Atrium", 1);
        List<String> names = roomManager.getAllRoomNames();
        List<String> loadedRooms = roomManager.getAllLoadedRoomnames();
        for (String nme : names) {
            if (!loadedRooms.contains(nme))
            {
                roomManager.loadRoom(nme);
            }
        }
    } catch (RemoteException e) {
        log.log(Level.SEVERE, e.toString(), e);
    }
}

/**
 * View all rooms connectionslog.
 *
 * @return
 */
public ConnectionsLog viewAllRooms() {
    try {
        return roomManager.returnRooms();
    } catch (RemoteException e) {
        log.log(Level.SEVERE, e.toString(), e);
    }
    return null;
}

/**
 * Verifies a username + password and under a certain number of
 * restrictions. If the user is verified then a new interface is created and
 * added to the registry.
 *
 * @param username
 * @param password
 * @return
 * @throws ConnectException
```

```
 */
public boolean login(String username, String password) throws ConnectException {
   Account returnedAccount = null;
   if (accountManager.verifyUser(username, password) && !bannedUsers.contains(username) && !connections.contains(username)) {
      returnedAccount = accountManager.getUserByName(username);
      if (returnedAccount != null) {
         addNewInterface(returnedAccount);
         log.log(Level.INFO, "User: {0} has logged in", username);
         return true;
      }
   }
   return false;
}

/**
 * Adds a new interface for the account specified, if the account given is
 * an admin, then it is assigned an AdminInterface object instead of a
 * classic UserInterface. It will then add this UI onto the registry.
 *
 * @param returnedAccount
 * @throws ConnectException
 */
public void addNewInterface(Account returnedAccount) throws ConnectException {
   try {
      String username = returnedAccount.getUsername();
      if (isAUser(username)) {
         IUserInterface newUI;
         if (isAAdmin(username)) {
            newUI = new AdminInterfaceImpl(returnedAccount, this);
         } else {
            newUI = new UserInterfaceImpl(returnedAccount);
         }
         newUI.connect();
         newUI.updateConnections(roomManager.returnRooms());
         newUI.setFrontpage(frontpage.getFrontpage());
         Registry registry = LocateRegistry.getRegistry(9807);
         registry.rebind(username, newUI);
         connections.put(username, newUI);
      } else {
      }
   } catch (RemoteException e) {
      log.log(Level.SEVERE, e.toString(), e);
   }
}

/**
 * Adds a username to be given an admin interface to the ArrayList.
 *
 * @param username
 */
public void addAdmin(String username) {
   serverAdmins.add(username);
   log.log(Level.INFO, "A new Admin has been added: {0}", username);
}

/**
```

```
 * Creates a new account, using the accountmanager to persist.
 *
 * @param username
 * @param password
 * @throws ConnectException
 */
public void createAccount(String username, String password) throws ConnectException {
   if (username.length() >= 5 && password.length() >= 5) {
      accountManager.addAccount(username, password);
      log.log(Level.INFO, "A new account has been created with the username: {0}", username);
   }
}

public String generateNewSessionKey() {
   SecurityHandler sh = new SecurityHandler();
   String filename = null;
   int keyIndex = 1;
   try {
      keyIndex = sh.generateSecretKey();
   } catch (NoSuchAlgorithmException ex) {
      Logger.getLogger(ServerController.class.getName()).log(Level.SEVERE, null, ex);
   }
   String paddedId = String.format("%05d", keyIndex);
   String secretKey = sh.getKey(keyIndex);
   String concatedString = secretKey + paddedId;
   try {
      filename = "secretkey" + clientKeysGenerated++ + ".txt";
      FileOutputStream fos = new FileOutputStream(new File(filename));
      fos.write(concatedString.getBytes());
      fos.flush();
      fos.close();
   } catch (FileNotFoundException ex) {
      Logger.getLogger(ServerController.class.getName()).log(Level.SEVERE, null, ex);
   } catch (IOException ex) {
      Logger.getLogger(ServerController.class.getName()).log(Level.SEVERE, null, ex);
   }
   return filename;
}

/**
 * Returns a List collection of all connected usernames, for more high level
 * interactions with the connected userbank. Used for Admin control.
 *
 * @return
 */
public List<String> getAllConnectedUsernames() {
   return new ArrayList(connections.keySet());
}

/**
 * Stops the server, and disconnects all connected users.
 */
public void stopServer() {
   serverManager.stopServer();
   //roomManager.stopRooms();
   for (String activename : connections.keySet()) {
```

```
            disconnectUser(activename);
        //    try {
         //       UnicastRemoteObject.unexportObject(connections.get(activename), true);
        //    } catch (NoSuchObjectException ex) {
        //        Logger.getLogger(ServerController.class.getName()).log(Level.SEVERE, null, ex);
        //    }
        }
        open = false;
        performance.close();
        pool.shutdown();
        //Registry reg;
        //try {
            //reg = LocateRegistry.getRegistry();
            //String[] regnames = reg.list();
            //for (String name : regnames) {
            //    System.out.println(name);
            //}
       // } catch (RemoteException ex) {
        //    Logger.getLogger(ServerController.class.getName()).log(Level.SEVERE, null, ex);
        // }
    }

    /**
     * Alert a user, this is used during Admin messages, as well as room private
     * whispers.
     *
     * @param msg
     * @param username
     */
    public void alertUser(Message msg, String username) {
        IUserInterface user = connections.get(username);
        try {
            user.recievePrivateMessage(msg);

        } catch (RemoteException ex) {
            Logger.getLogger(ServerController.class
                    .getName()).log(Level.SEVERE, null, ex);
        }
    }

    /**
     * Checks to see if the user is logged in, used during the Login process.
     *
     * @param username
     * @return
     */
    public boolean isUserLoggedIn(String username) {
        boolean loggedIn = false;
        if (connections.containsKey(username)) {
            loggedIn = true;
        }
        return loggedIn;
    }

    /**
     * Bans a user from the server, by a username. Will send them a private
```

```
     * message too.
     *
     * @param username
     */
    public void banUser(String username) {
        if (connections.containsKey(username)) {
            IUserInterface ui = connections.get(username);
            try {
                ui.recievePrivateMessage(new Message("System", username, "You have been banned from the server", 3));
                ui.disconnect();
            } catch (RemoteException ex) {
            }
            bannedUsers.add(username);
        }
    }

}
```

UserInterfaceImpl Class:

```
package conclave.controllers;

import conclave.ConclaveHandlers.RoomManager;
import conclave.db.Account;
import conclaveinterfaces.IConclaveRoom;
import conclaveinterfaces.IConferenceRoom;
import conclaveinterfaces.IUserInterface;
import java.awt.Dimension;
import java.net.InetSocketAddress;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import model.Announcement;
import model.Chatlog;
import model.ConnectionsLog;
import model.Message;
import model.ServerFrontpage;

/**
 * User interface interacts with the server with general interactions Also is
 * responsible for providing state updates and view information. Consider this
 * the Controller in the MVC paradigm.
 *
 * @author BradleyW
 */
public class UserInterfaceImpl extends UnicastRemoteObject implements IUserInterface {

    //Model variables used to model the state
    private Account account;
    private ConnectionsLog connectionsLog;
    private IConclaveRoom activeRoom;
```

```
private RoomManager roomListingsRoom;
private final Chatlog chatLog;
private ServerFrontpage ownFrontpage;

//These state flags represent Server and Room state changes.
//These are primarilly used by the GUI to build an updated view.
private int lastMessageLine;
private boolean inRoom;
private boolean connected;
private boolean connectionsUpdate;
private boolean frontpageUpdated;
private boolean activeWebcamUpdated;

public UserInterfaceImpl(Account account) throws RemoteException {
   lastMessageLine = 0;
   chatLog = new Chatlog();
   this.account = account;
   connected = false;
   inRoom = false;
   connectionsUpdate = true;
   connectionsLog = new ConnectionsLog();
   roomListingsRoom = RoomManager.getInstance();
   ownFrontpage = new ServerFrontpage();
   frontpageUpdated = true;
}

@Override
public String getActiveRoomName() throws RemoteException {
   if (inRoom) {
      return activeRoom.getRoomName();
   } else {
      return null;
   }
}

@Override
public boolean hasPassword(String entryname) throws RemoteException {
   return roomListingsRoom.hasPassword(entryname);
}

@Override
public boolean joinRoom(String entryName, String password) throws RemoteException {
   boolean ok = false;
   if (roomListingsRoom.valdiateRoom(entryName, password)) {
      ok = joinRoom(entryName);
      inRoom = true;
   }
   return ok;
}

@Override
public boolean joinRoom(String entryName) throws RemoteException {
   boolean ok = false;
   try {
      if (connected && !inRoom) {
         final Registry registry = LocateRegistry.getRegistry(9807);
```

```
            activeRoom = (IConclaveRoom) registry.lookup(entryName);
            String username = account.getUsername();
            if (activeRoom.isOnline()) {
               activeRoom.addUser(username, this);
               connectionsLog = activeRoom.getAllConnections();
               inRoom = true;
               ok = true;
               connectionsUpdate = true;
               if (activeRoom.getType() == 2) {
                  activeWebcamUpdated = true;
               }
            }
         }
      } catch (NotBoundException e) {
         e.printStackTrace();
      }
      return ok;
   }

   @Override
   public boolean inRoom() throws RemoteException {
      return inRoom;
   }

   @Override
   public void leaveRoom() throws RemoteException {
      if (inRoom && connected) {
         String username = account.getUsername();
         //String roomname = activeRoom.getRoomName();
         if (activeRoom.getType() == 2) {
            stopBroadcasting();
         }
         activeRoom.removeUser(username);
         activeRoom = null;
         inRoom = false;
         connectionsLog = roomListingsRoom.returnRooms();
         connectionsUpdate = true;
         frontpageUpdated = true;
      }
   }

   @Override
   public void stopBroadcasting() {
      try {
         if (activeRoom.getType() == 2) {
            String username = account.getUsername();
            IConferenceRoom confRoom = (IConferenceRoom) activeRoom;
            if (confRoom.getStreamerName() != null) {
               if (confRoom.getStreamerName().equals(username)) {
                  confRoom.stopBroadcasting(username);
               }
            }
         }
      } catch (RemoteException ex) {
         Logger.getLogger(UserInterfaceImpl.class.getName()).log(Level.SEVERE, null, ex);
      }
```

```
}

@Override
public void postMessage(String message) throws RemoteException {
   if (inRoom && connected) {
      String username = getUsername();
      String roomId = activeRoom.getRoomName();
      Message newMessage;
      newMessage = new Message(username, roomId, message, 1);
      activeRoom.postMessage(newMessage);
   }
}

@Override
public void updateChatLog(Message message) throws RemoteException {
   if (message != null) {
      chatLog.addMessage(message);
      lastMessageLine++;
   } else {
      System.out.println("Message is null.");
   }
}

@Override
public void updateConnections(ConnectionsLog newLog) throws RemoteException {
   connectionsLog = newLog;
   connectionsUpdate = true;
}

@Override
public ConnectionsLog viewAllConnections() throws RemoteException {
   connectionsUpdate = false;
   if (!inRoom) {
      return roomListingsRoom.returnRooms();
   } else {
      return activeRoom.getAllConnections();
   }
}

@Override
public void sendPrivateMessage(String message, String recipientID) throws RemoteException {
   if (connected && inRoom) {
      Message privateSentMessage = new Message(account.getUsername(), recipientID, message, 4);
      activeRoom.whisper(privateSentMessage);
   }
}

@Override
public void recievePrivateMessage(Message message) throws RemoteException {
   chatLog.addMessage(message);
   lastMessageLine++;
}

@Override
public void leaveServer() throws RemoteException {
   if (inRoom) {
```

```
            leaveRoom();
        }
        disconnect();
    }

    @Override
    public boolean isConnected() throws RemoteException {
        return connected;
    }

    @Override
    public void connect() throws RemoteException {
        connected = true;
    }

    @Override
    public void disconnect() throws RemoteException {
        connected = false;
    }

    @Override
    public String getUsername() throws RemoteException {
        return account.getUsername();
    }

    @Override
    public String exportChatLog() {
        return chatLog.viewEntries();
    }

    @Override
    public LinkedList<Message> getChatlogUpdates(int lstMsgRecieved) throws RemoteException {
        LinkedList<Message> returnArray = chatLog.getAllEntriesAfter(lstMsgRecieved);
        return returnArray;
    }

    @Override
    public int getLastMessageLine() throws RemoteException {
        return lastMessageLine;
    }

    @Override
    public boolean hasConnectionsUpdated() throws RemoteException {
        return connectionsUpdate;
    }

    @Override
    public int getRoomType() throws RemoteException {
        return activeRoom.getType();
    }

    @Override
    public int getType() throws RemoteException {
        return 1;
    }
```

```
@Override
public List<Announcement> getFrontpage() throws RemoteException {
   frontpageUpdated = false;
   return new ArrayList(ownFrontpage.getFrontpage());
}

@Override
public void setFrontpage(List<Announcement> frontPage) throws RemoteException {
   ownFrontpage.setFrontpage(frontPage);
   if (!inRoom) {
      frontpageUpdated = true;
   }
}

@Override
public void updateFrontpage(String username, String announcment) throws RemoteException {
   ownFrontpage.addNewAnnouncment(username, announcment);
   if (!inRoom) {
      frontpageUpdated = true;
   }
}

@Override
public boolean hasFrontpageUpdated() throws RemoteException {
   return frontpageUpdated;
}

@Override
public Dimension getConferenceDimension() throws RemoteException {
   Dimension returnDimension = null;
   if (activeRoom.getType() == 2) {
      IConferenceRoom conference = (IConferenceRoom) activeRoom;
      returnDimension = conference.getDimension();
   }
   return returnDimension;
}

@Override
public boolean isConferenceStreaming() throws RemoteException {
   boolean isStreaming = false;
   if (activeRoom.getType() == 2) {
      IConferenceRoom conference = (IConferenceRoom) activeRoom;
      if (conference.isStreaming()) {
         isStreaming = true;
      }
   }
   return isStreaming;
}

@Override
public InetSocketAddress getStreamerLocation() throws RemoteException {
   if (activeRoom.getType() == 2) {
      IConferenceRoom conference = (IConferenceRoom) activeRoom;
      return conference.getStreamerIp();
   }
   return null;
```

```
    }

    @Override
    public void broadcastToConference(InetSocketAddress loc, Dimension d) throws RemoteException {
        if (activeRoom.getType() == 2) {
            IConferenceRoom conference = (IConferenceRoom) activeRoom;
            if (!conference.isStreaming()) {
                conference.startBroadcasting(getUsername(), loc, d);
                activeWebcamUpdated = true;
            }
        }
    }

    @Override
    public boolean hasStreamerUpdated() throws RemoteException {
        boolean hasUpdated = activeWebcamUpdated;
        activeWebcamUpdated = false;
        return hasUpdated;
    }

    @Override
    public String getStreamerName() throws RemoteException {
        String name = null;
        if (activeRoom.getType() == 2) {
            IConferenceRoom conference = (IConferenceRoom) activeRoom;
            if (conference.isStreaming()) {
                name = conference.getStreamerName();
            }
        }
        return name;
    }

    @Override
    public void updateStreamer() throws RemoteException {
        activeWebcamUpdated = true;
    }
}
```

Account Class:

```
package conclave.db;

import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Cacheable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Lob;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.xml.bind.annotation.XmlRootElement;

/**
 *
```

```
 * @author BradleyW
 */
@Entity
@Table(name = "ACCOUNT")
@XmlRootElement
@Cacheable(true)
@NamedQueries({
   @NamedQuery(name = "Account.findAll", query = "SELECT a FROM Account a"),
   @NamedQuery(name = "Account.findByUserid", query = "SELECT a FROM Account a WHERE a.userid = :userid"),
   @NamedQuery(name = "Account.findByUsername", query = "SELECT a FROM Account a WHERE a.username = :username"),
   @NamedQuery(name = "Account.findByHashedpassword", query = "SELECT a FROM Account a WHERE a.hashedpassword =
:hashedpassword")})
public class Account implements Serializable {

   private static final long serialVersionUID = 1L;
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Basic(optional = false)
   @Column(name = "USERID")
   private Integer userid;
   @Basic(optional = false)
   @Column(name = "USERNAME")
   private String username;
   @Basic(optional = false)
   @Column(name = "HASHEDPASSWORD")
   private String hashedpassword;
   @Basic(optional = false)
   @Lob
   @Column(name = "SALT")
   private Serializable salt;

   public Account() {
   }

   public Account(Integer userid) {
      this.userid = userid;
   }

   public Account(Integer userid, String username, String hashedpassword, Serializable salt) {
      this.userid = userid;
      this.username = username;
      this.hashedpassword = hashedpassword;
      this.salt = salt;
   }

   public Integer getUserid() {
      return userid;
   }

   public void setUserid(Integer userid) {
      this.userid = userid;
   }

   public String getUsername() {
      return username;
   }
```

```
public void setUsername(String username) {
   this.username = username;
}

public String getHashedpassword() {
   return hashedpassword;
}

public void setHashedpassword(String hashedpassword) {
   this.hashedpassword = hashedpassword;
}

public byte[] getSalt()
{
   String s = (String) salt;
   int len = s.length();
   byte[] data = new byte[len / 2];
   for (int i = 0; i < len; i += 2) {
   data[i / 2] = (byte) ((Character.digit(s.charAt(i), 16) << 4)
                + Character.digit(s.charAt(i+1), 16));
}
   return data;
}

public void setSalt(byte[] saltBytes)
{
char[] hexChars = new char[saltBytes.length * 2];
char[] hexArray = "0123456789ABCDEF".toCharArray();
for ( int j = 0; j < saltBytes.length; j++ ) {
   int v = saltBytes[j] & 0xFF;
   hexChars[j * 2] = hexArray[v >>> 4];
   hexChars[j * 2 + 1] = hexArray[v & 0x0F];
 }
this.salt = new String(hexChars);
}

@Override
public int hashCode() {
   int hash = 0;
   hash += (userid != null ? userid.hashCode() : 0);
   return hash;
}

@Override
public boolean equals(Object object) {
   // TODO: Warning - this method won't work in the case the id fields are not set
   if (!(object instanceof Account)) {
      return false;
   }
   Account other = (Account) object;
   if ((this.userid == null && other.userid != null) || (this.userid != null && !this.userid.equals(other.userid))) {
      return false;
   }
   return true;
}
```

```
    @Override
    public String toString() {
        return "conclave.db.Account[ userid=" + userid + " ]";
    }

}
```

Room Class:

```
package conclave.db;

import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Lob;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.xml.bind.annotation.XmlRootElement;

/**
 *
 * @author BradleyW
 */
@Entity
@Table(name = "ROOM")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "Room.findAll", query = "SELECT r FROM Room r"),
    @NamedQuery(name = "Room.findByRoomid", query = "SELECT r FROM Room r WHERE r.roomid = :roomid"),
    @NamedQuery(name = "Room.findByRoomname", query = "SELECT r FROM Room r WHERE r.roomname = :roomname"),
    @NamedQuery(name = "Room.findByHashedpassword", query = "SELECT r FROM Room r WHERE r.hashedpassword =
:hashedpassword"),
    @NamedQuery(name = "Room.findByRoomtype", query = "SELECT r FROM Room r WHERE r.roomtype = :roomtype")})
public class Room implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "ROOMID")
    private Integer roomid;
    @Basic(optional = false)
    @Column(name = "ROOMNAME")
    private String roomname;
    @Basic(optional = false)
    @Column(name = "HASHEDPASSWORD")
    private String hashedpassword;
    @Basic(optional = false)
    @Lob
    @Column(name = "SALT")
    private Serializable salt;
    @Basic(optional = false)
```

```
@Column(name = "ROOMTYPE")
private String roomtype;

public Room() {
}

public Room(Integer roomid) {
   this.roomid = roomid;
}

public Room(Integer roomid, String roomname, String hashedpassword, Serializable salt, String roomtype) {
   this.roomid = roomid;
   this.roomname = roomname;
   this.hashedpassword = hashedpassword;
   this.salt = salt;
   this.roomtype = roomtype;
}

public Integer getRoomid() {
   return roomid;
}

public void setRoomid(Integer roomid) {
   this.roomid = roomid;
}

public String getRoomname() {
   return roomname;
}

public void setRoomname(String roomname) {
   this.roomname = roomname;
}

public String getHashedpassword() {
   return hashedpassword;
}

public void setHashedpassword(String hashedpassword) {
   this.hashedpassword = hashedpassword;
}

public byte[] getSalt()
{
   String s = (String) salt;
   int len = s.length();
   byte[] data = new byte[len / 2];
   for (int i = 0; i < len; i += 2) {
   data[i / 2] = (byte) ((Character.digit(s.charAt(i), 16) << 4)
                + Character.digit(s.charAt(i+1), 16));
}
   return data;
}

public void setSalt(byte[] saltBytes)
{
```

```
  char[] hexChars = new char[saltBytes.length * 2];
  char[] hexArray = "0123456789ABCDEF".toCharArray();
  for ( int j = 0; j < saltBytes.length; j++ ) {
     int v = saltBytes[j] & 0xFF;
     hexChars[j * 2] = hexArray[v >>> 4];
     hexChars[j * 2 + 1] = hexArray[v & 0x0F];
   }
  this.salt = new String(hexChars);
  }

  public String getRoomtype() {
     return roomtype;
  }

  public void setRoomtype(String roomtype) {
     this.roomtype = roomtype;
  }

  @Override
  public int hashCode() {
     int hash = 0;
     hash += (roomid != null ? roomid.hashCode() : 0);
     return hash;
  }

  @Override
  public boolean equals(Object object) {
     // TODO: Warning - this method won't work in the case the id fields are not set
     if (!(object instanceof Room)) {
        return false;
     }
     Room other = (Room) object;
     if ((this.roomid == null && other.roomid != null) || (this.roomid != null && !this.roomid.equals(other.roomid))) {
        return false;
     }
     return true;
  }

  @Override
  public String toString() {
     return "conclave.db.Room[ roomid=" + roomid + " ]";
  }

}
```

Sessionkeys Class:

```
package conclave.db;

import java.io.Serializable;
import java.util.Collection;
import javax.persistence.Basic;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
```

```
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlTransient;

/**
 *
 * @author BradleyW
 */
@Entity
@Table(name = "SESSIONKEYS")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "Sessionkeys.findAll", query = "SELECT s FROM Sessionkeys s"),
    @NamedQuery(name = "Sessionkeys.findByKeyidentify", query = "SELECT s FROM Sessionkeys s WHERE s.keyidentify = :keyidentify"),
    @NamedQuery(name = "Sessionkeys.findByKeystring", query = "SELECT s FROM Sessionkeys s WHERE s.keystring = :keystring")})
public class Sessionkeys implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "KEYIDENTIFY")
    private Integer keyidentify;
    @Basic(optional = false)
    @Column(name = "KEYSTRING")
    private String keystring;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "keyid")
    private Collection<Userkeys> userkeysCollection;

    public Sessionkeys() {
    }

    public Sessionkeys(Integer keyidentify) {
        this.keyidentify = keyidentify;
    }

    public Sessionkeys(Integer keyidentify, String keystring) {
        this.keyidentify = keyidentify;
        this.keystring = keystring;
    }

    public Integer getKeyidentify() {
        return keyidentify;
    }

    public void setKeyidentify(Integer keyidentify) {
        this.keyidentify = keyidentify;
    }

    public String getKeystring() {
        return keystring;
    }
```

```
    public void setKeystring(String keystring) {
        this.keystring = keystring;
    }

    @XmlTransient
    public Collection<Userkeys> getUserkeysCollection() {
        return userkeysCollection;
    }

    public void setUserkeysCollection(Collection<Userkeys> userkeysCollection) {
        this.userkeysCollection = userkeysCollection;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (keyidentify != null ? keyidentify.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id fields are not set
        if (!(object instanceof Sessionkeys)) {
            return false;
        }
        Sessionkeys other = (Sessionkeys) object;
        if ((this.keyidentify == null && other.keyidentify != null) || (this.keyidentify != null && !this.keyidentify.equals(other.keyidentify))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "conclave.db.Sessionkeys[ keyidentify=" + keyidentify + " ]";
    }

}
```

Userkeys class:

```
package conclave.db;

import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.xml.bind.annotation.XmlRootElement;
```

```
/**
 *
 * @author BradleyW
 */
@Entity
@Table(name = "USERKEYS")
@XmlRootElement
@NamedQueries({
   @NamedQuery(name = "Userkeys.findAll", query = "SELECT u FROM Userkeys u"),
   @NamedQuery(name = "Userkeys.findByUserkeyid", query = "SELECT u FROM Userkeys u WHERE u.userkeyid = :userkeyid"),
   @NamedQuery(name = "Userkeys.findByUsername", query = "SELECT u FROM Userkeys u WHERE u.username = :username"),
   @NamedQuery(name = "Userkeys.findByKeyId", query = "SELECT u FROM Userkeys u WHERE u.keyid = :keyid")})
public class Userkeys implements Serializable {

   private static final long serialVersionUID = 1L;
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Basic(optional = false)
   @Column(name = "USERKEYID")
   private Integer userkeyid;
   @Basic(optional = false)
   @Column(name = "USERNAME")
   private String username;
   @JoinColumn(name = "KEYID", referencedColumnName = "KEYIDENTIFY")
   @ManyToOne(optional = false)
   private Sessionkeys keyid;

   public Userkeys() {
   }

   public Userkeys(Integer userkeyid) {
      this.userkeyid = userkeyid;
   }

   public Userkeys(Integer userkeyid, String username) {
      this.userkeyid = userkeyid;
      this.username = username;
   }

   public Integer getUserkeyid() {
      return userkeyid;
   }

   public void setUserkeyid(Integer userkeyid) {
      this.userkeyid = userkeyid;
   }

   public String getUsername() {
      return username;
   }

   public void setUsername(String username) {
      this.username = username;
   }
```

```
    public Sessionkeys getKeyid() {
        return keyid;
    }

    public void setKeyid(Sessionkeys keyid) {
        this.keyid = keyid;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (userkeyid != null ? userkeyid.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id fields are not set
        if (!(object instanceof Userkeys)) {
            return false;
        }
        Userkeys other = (Userkeys) object;
        if ((this.userkeyid == null && other.userkeyid != null) || (this.userkeyid != null && !this.userkeyid.equals(other.userkeyid))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "conclave.db.Userkeys[ userkeyid=" + userkeyid + " ]";
    }

}
```

ConferenceRoom class:

```
package conclave.model;

import conclaveinterfaces.IConferenceRoom;
import conclaveinterfaces.IUserInterface;
import model.Message;
import java.awt.Dimension;
import java.net.InetSocketAddress;
import java.rmi.RemoteException;
import java.util.logging.Level;
import java.util.logging.Logger;

/**Conference room will host Conclave Conference rooms, this room is different from textroom in that
 * it provides an IP and Socket of the streamer; as well as a Dimension. It extends a text room as
 * it provides the same basic functionality but adds to it.
 *
 * @author BradleyW
 */
public class ConferenceRoom extends TextRoom implements IConferenceRoom {

    //Streamer Variables.
    private String streamerName; // streamer name
```

```
private boolean streaming; //is a user currently streaming
private Dimension streamingDimension; //streaming dimension
private InetSocketAddress streamerIp; //location of streaming network

//Logger.
private static final Logger log = Logger.getLogger(ConferenceRoom.class.getName());

/**
 * Conference Room is initiated with a name, like a textroom.
 * @param iroomName
 * @throws RemoteException
 */
public ConferenceRoom(String iroomName) throws RemoteException {
   super(iroomName);
   this.roomType = 2; //The type is 2, TextRoom is 1.
   streaming = false;
   streamerName = "";
}

/**
 * Returns the streamer socketIP of the current broadcaster.
 * @return
 * @throws RemoteException
 */
@Override
public InetSocketAddress getStreamerIp() throws RemoteException {
   if (streaming) {
      return streamerIp;
   }
   return null;
}

/**
 * This is used by a user to start streaming to a room, it sets a couple of variables for listeners to use
 * to recieve the video stream. Muted users cannot stream, however.
 * @param username
 * @param networkloc
 * @param d
 * @throws RemoteException
 */
@Override
public void startBroadcasting(String username, InetSocketAddress networkloc, Dimension d) throws RemoteException {
   if (censorList.contains(username)) {
      whisper(new Message(roomName, username, "Muted users cannot stream.", 2));
   } else if (!streaming) {
      this.streamerIp = networkloc;
      this.streamerName = username;
      this.streamingDimension = d;
      this.streaming = true;
      updateActiveListeners();
      log.log(Level.INFO, "User: {0} has started to broadcast at room: {1}", new Object[]{username, roomName});
   }
}

/**
 * Updates all the streamer updated flag, this forces the view to update the streamer panel.
```

```
 * Used for stopping or starting a broadcast.
 * @throws RemoteException
 */
public void updateActiveListeners() throws RemoteException {
   for (IUserInterface ui : roomConnections.values()) {
      ui.updateStreamer();
   }
}

/**
 * Stops a broadcast, by setting all the streamer variables to null.
 * @param streamerName
 * @throws RemoteException
 */
@Override
public void stopBroadcasting(String istreamerName) throws RemoteException {
   if (istreamerName.equals(this.streamerName)) {
      streamingDimension = null;
      this.streamerName = null;
      streamerIp = null;
      streaming = false;
      updateActiveListeners();
      log.log(Level.INFO, "User: {0} has stopped broadcasting at room: {1}", new Object[]{streamerName, roomName});
   }
}

/**
 * Gets a info view, this is used to display connection entries.
 * @return
 * @throws RemoteException
 */
@Override
public String getInfo() throws RemoteException {
   return "[ConferenceRoom] " + currentConnections + "/" + roomLimit + " {" + online + "}";
}

/**
 * Returns if the room currently has an active streamer.
 * @return
 * @throws RemoteException
 */
@Override
public boolean isStreaming() throws RemoteException {
   return streaming;
}

/**
 * Returns the dimension of the currently active stream, used for listeners
 * to know how wide to paint a panel.
 * @return
 * @throws RemoteException
 */
@Override
public Dimension getDimension() throws RemoteException {
   if (streaming) {
      return streamingDimension;
```

```
        }
        return null;
    }

    /**
     * Returns the current streamer name, used for the view.
     * @return
     * @throws RemoteException
     */
    @Override
    public String getStreamerName() throws RemoteException {
        if (streaming) {
            return streamerName;
        }
        return null;
    }
}
```

TextRoom class:

```
package conclave.model;

import model.ConnectionsLog;
import model.Message;
import java.io.IOException;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.logging.Level;
import java.util.logging.Logger;
import conclaveinterfaces.IConclaveRoom;
import conclaveinterfaces.IUserInterface;

/**
 *TextRoom, provides the basic room interactions. All other rooms should extend this,
 * as it provides ChatLog + ConnectionLog functionality.
 * @author BradleyW
 */
public class TextRoom extends UnicastRemoteObject implements IConclaveRoom {

    //Basic room variables.
    public String roomName;
    public int roomLimit;
    public int currentConnections;

    public HashMap<String, IUserInterface> roomConnections;
    public ConnectionsLog connectionsLog;
    public ArrayList<String> censorList;

    public boolean online;
    public int roomType;

    private final Logger log = Logger.getLogger(TextRoom.class.getName());

    public TextRoom(String iroomName) throws RemoteException{
        this.roomName = iroomName;
        this.online = true;
```

```
    roomConnections = new HashMap<>();
    censorList = new ArrayList<>();
    connectionsLog = new ConnectionsLog();
    roomLimit = 20;
    roomType = 1;
    currentConnections = 0;
}

/**
 * Stops a room, preventing new connections and then kicking everyone from the room.
 * @throws RemoteException
 */
@Override
public void stopRoom() throws RemoteException
{
    online = false;
    System.out.println("Stopping room.");
    for (String username : roomConnections.keySet())
    {
        removeUser(username);
    }
}

/**
 * Adds a user to the room, used when joining a room.
 * @param username
 * @param user
 * @throws RemoteException
 */
@Override
public void addUser(String username, IUserInterface user) throws RemoteException
{
    if (currentConnections < roomLimit)
    {
        roomConnections.put(username, user);
        connectionsLog.addConnection(username, "User");
        String msg = "You have joined room: " + roomName;
        Message welcomeMessage = new Message(roomName, username, msg, 2);
        whisper(welcomeMessage);
        updateAllClientsConnections();
        log.log(Level.INFO, "User: {0} has joined the room: {1}", new Object[] {username, roomName});
        currentConnections++;
    }
}

/**
 * Removes a user from the room, used when kicking or leaving a room.
 * @param username
 * @throws RemoteException
 */
@Override
public void removeUser(String username) throws RemoteException
{
    String msg = "You have been removed from the room";
    Message removeMessage = new Message(roomName, username, msg, 2);
    IUserInterface ui = roomConnections.get(username);
```

```
    if (ui!=null)
    {
        ui.updateChatLog(removeMessage);
    }
    roomConnections.remove(username);
    connectionsLog.removeConnection(username);
    updateAllClientsConnections();
    log.log(Level.INFO, "User: {0} has been removed from the room: {1}", new Object[] {username, roomName});
    currentConnections--;
}

/**
 * Posts a chatlog message to all users in the room, if they are not muted.
 *
 * @param message
 * @throws RemoteException
 */
@Override
public void postMessage(Message message)throws RemoteException
{
    if (!censorList.contains(message.getSenderName()))
    {
        updateAllClientsChatlog(message);
    } else {
        whisper(new Message(roomName, message.getSenderName(), "You are currently muted", 2));
    }
}
/**
 * Edit the roomlimit of a room. No longer used in implementation.
 * @param limit
 * @throws RemoteException
 */
@Override
public void setLimit(int limit)throws RemoteException
{
    this.roomLimit = limit;
}

/**
 * Updates all the chatlogs of users with the message, used
 * during chatlog updates, or system messages.
 *
 * @param msg
 * @throws RemoteException
 */
@Override
public void updateAllClientsChatlog(Message msg) throws RemoteException
{
        for (IUserInterface ui : roomConnections.values())
        {
            try {
                ui.updateChatLog(msg);
            } catch (IOException e)
            {
                String username = ui.getUsername();
                removeUser(username);
```

```
            updateAllClientsConnections();
            e.printStackTrace();
        }
      }
}

/**
 * Returns the roomname.
 *
 * @return
 * @throws RemoteException
 */
@Override
public String getRoomName() throws RemoteException{
   return roomName;
}

/**
 * Returns the roomlimit.
 *
 * @return
 * @throws RemoteException
 */
@Override
public int getRoomLimit() throws RemoteException{
   return roomLimit;
}

/**
 * Returns a boolean based on if the room is open/closed.
 * @return
 * @throws RemoteException
 */
@Override
public boolean isOnline() throws RemoteException{
   return online;
}

/**
 * Returns a connection info, used to display the room in the connections log.
 * This should be overridden by any future implementations of rooms.
 * @return
 * @throws RemoteException
 */
@Override
public String getInfo() throws RemoteException {
   return "[TextRoom] " + currentConnections + "/" + roomLimit + " {" + online + "}";
}

/**
 * Returns a connectionlog of all users in the room, used when joining a room to get
 * all the users.
 *
 * @return
 * @throws RemoteException
 */
```

```
@Override
public ConnectionsLog getAllConnections() throws RemoteException
{
   return connectionsLog;
}

/**
 * Updates all clients connections, used when removing a player or a new player
 * joins the room.
 * @throws RemoteException
 */
@Override
public void updateAllClientsConnections() throws RemoteException
{
   for (IUserInterface ui : roomConnections.values())
   {
      try {
         ui.updateConnections(connectionsLog);
      } catch (RemoteException e)
      {
         removeUser(ui.getUsername()); //also as a failsafe, determines if the user is offline
      }
   }

}

/**
 * Mutes a user by adding them to the CensorList.
 * @param username
 * @throws RemoteException
 */
@Override
public void addCensoredUser(String username) throws RemoteException
{
   censorList.add(username);
}

/**
 * Unmutes a user by removing them from the censor list.
 * @param username
 * @throws RemoteException
 */
@Override
public void uncensorUser(String username) throws RemoteException
{
   if (censorList.contains(username))
   {
      censorList.remove(username);
   }
}

/**
 * Whispers a user with a private message.
 * @param msg
 * @throws RemoteException
 */
```

```
@Override
public void whisper(Message msg) throws RemoteException
{
   String recipientUsername = msg.getRecipientId();
   IUserInterface ui = roomConnections.get(recipientUsername);
   ui.updateChatLog(msg);
}


/**
 * Kicks the user from the room.
 * @param username
 * @throws RemoteException
 */
@Override
public void kickUser(String username) throws RemoteException
{
   IUserInterface ui = roomConnections.get(username);
   ui.leaveRoom(); //This calls the removeUser method, so no need to call it again.
}

/**returns the type, for textroom this is 1.
 *
 * @return
 * @throws RemoteException
 */
@Override
public int getType() throws RemoteException
{
   return roomType;
}

/**
 * Does the room contain this username?
 *
 * @param username
 * @return
 * @throws RemoteException
 */
@Override
public boolean hasUser(String username) throws RemoteException {
   boolean has = false;
   if (roomConnections.containsKey(username))
   {
      has = true;
   }
   return has;
}

/**
 * Closes a room to new connections, does not interfere with current connections
 *
 * @throws RemoteException
 */
@Override
public void closeRoom() throws RemoteException {
```

```
    online = false;
  }

  /**
   * Opens a room to new connections.
   *
   * @throws RemoteException
   */
  @Override
  public void openRoom() throws RemoteException {
    online = true;
  }
}
```

RMISecurityPolicyLoader class:

```
package conclave.rmiPolicy;

import java.net.URL;
import java.rmi.RMISecurityManager;

/**Acknowledgements: David Bowes
 *This class loads a RMI security policy.
 * @author dhb
 */
public class RMISecurityPolicyLoader {

  private static boolean loaded = false;

  /**
   * Loads a policy with the default name.
   */
  public static void LoadDefaultPolicy() {
    LoadPolicy("RMISecurity.policy");
  }

  /**
   * Loads a policy using the specified filename, then assigns a security manager
   * to the registry.
   * @param policy
   */
  public static void LoadPolicy(String policy) {
    if (!loaded) {
      loaded = true;
      ClassLoader cl = RMISecurityPolicyLoader.class.getClassLoader();
      URL url = cl.getResource(policy);
      if (url == null) {
        //Policy not found.
      } else {
        //policy found
      }
      System.setProperty("java.security.policy", url.toString());

      if (System.getSecurityManager() == null) {
        System.setSecurityManager(new RMISecurityManager());
      }
    }
  }
```

```
    private RMISecurityPolicyLoader() {
    }
}
```

RegistryLoader class:

```
package conclave.rmiPolicy;

import java.net.InetAddress;
import java.net.UnknownHostException;
import java.rmi.RemoteException;
/**This class create an RMI registry at port 9807.
 *
 * @author BradleyW
 */
public class RegistryLoader {

    private static boolean loaded = false;

    public static void Load() throws RemoteException, UnknownHostException {
        if (loaded) {
            return;
        }
        System.setProperty("java.rmi.server.hostname",InetAddress.getLocalHost().getHostAddress());
        java.rmi.registry.LocateRegistry.createRegistry(9807);
        loaded = true;
    }

    private RegistryLoader() {
    }
}
```

Encryptor class:

```
package util;

import java.nio.charset.Charset;
import java.security.GeneralSecurityException;
import java.security.InvalidKeyException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import javax.xml.bind.DatatypeConverter;

/**
 *
 * @author BradleyW
 */
public class Encryptor {

    public Encryptor() {
    }

    public static String encrypt(final String plainMessage,final String symKeyHex) {
```

```
    final byte[] symKeyData = DatatypeConverter.parseHexBinary(symKeyHex);
    final byte[] encodedMessage = plainMessage.getBytes(Charset .forName("UTF-8"));
    try {
      final Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
      final int blockSize = cipher.getBlockSize();
      final SecretKeySpec symKey = new SecretKeySpec(symKeyData, "AES");
      final byte[] ivData = new byte[blockSize];
      final SecureRandom rnd = SecureRandom.getInstance("SHA1PRNG");
      rnd.nextBytes(ivData);
      final IvParameterSpec iv = new IvParameterSpec(ivData);
      cipher.init(Cipher.ENCRYPT_MODE, symKey, iv);
      final byte[] encryptedMessage = cipher.doFinal(encodedMessage);
      final byte[] ivAndEncryptedMessage = new byte[ivData.length + encryptedMessage.length];
      System.arraycopy(ivData, 0, ivAndEncryptedMessage, 0, blockSize);
      System.arraycopy(encryptedMessage, 0, ivAndEncryptedMessage,blockSize, encryptedMessage.length);
      final String ivAndEncryptedMessageBase64 = DatatypeConverter.printBase64Binary(ivAndEncryptedMessage);
      return ivAndEncryptedMessageBase64;
    } catch (InvalidKeyException e) {
      throw new IllegalArgumentException(
          "key argument does not contain a valid AES key");
    } catch (GeneralSecurityException e) {
      throw new IllegalStateException(
          "Unexpected exception during encryption", e);
    }
  }

  public static String decrypt(final String ivAndEncryptedMessageBase64,
      final String symKeyHex) {
    final byte[] symKeyData = DatatypeConverter.parseHexBinary(symKeyHex);
    final byte[] ivAndEncryptedMessage = DatatypeConverter
        .parseBase64Binary(ivAndEncryptedMessageBase64);
    try {
      final Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
      final int blockSize = cipher.getBlockSize();
      final SecretKeySpec symKey = new SecretKeySpec(symKeyData, "AES");
      final byte[] ivData = new byte[blockSize];
      System.arraycopy(ivAndEncryptedMessage, 0, ivData, 0, blockSize);
      final IvParameterSpec iv = new IvParameterSpec(ivData);
      final byte[] encryptedMessage = new byte[ivAndEncryptedMessage.length
          - blockSize];
      System.arraycopy(ivAndEncryptedMessage, blockSize,
          encryptedMessage, 0, encryptedMessage.length);
      cipher.init(Cipher.DECRYPT_MODE, symKey, iv);
      final byte[] encodedMessage = cipher.doFinal(encryptedMessage);
      final String message = new String(encodedMessage,
          Charset.forName("UTF-8"));
      return message;
    } catch (InvalidKeyException e) {
      throw new IllegalArgumentException(
          "key argument does not contain a valid AES key");
    } catch (GeneralSecurityException e) {
      throw new IllegalStateException(
          "Unexpected exception during decryption", e);
    }
  }
```

```
  public static String hashPassword(String password, byte[] salt) {
    String hashedString = null;
    try {
      MessageDigest md = MessageDigest.getInstance("SHA-256");
      md.update(salt);
      byte[] hashbytes = md.digest(password.getBytes());
      StringBuilder sb = new StringBuilder();
      for (int i = 0; i < hashbytes.length; i++) {
        sb.append(Integer.toString((hashbytes[i] & 0xff) + 0x100, 16).substring(1));
      }
      hashedString = sb.toString();
    } catch (NoSuchAlgorithmException e) {
      e.printStackTrace();
    }
    return hashedString;
  }
}
```

PerformanceLogger class:

```
package util;

import com.sun.management.OperatingSystemMXBean;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.lang.management.ManagementFactory;
import java.net.InetAddress;
import java.net.Socket;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.management.MBeanServerConnection;

/**
 * This class is responsible for writing performance information into a .csv
 * file, for use in load testing and performance analysis.
 *
 * @author BradleyW
 */
public class PerformanceLogger {

  PrintWriter writer; //Printwriter used to write to a txt file.
  private Socket sock; //Socket used to find response times.

  private InetAddress ip; //Server IP.
  private int port; //Server port.

  MBeanServerConnection mbsc;

  /**
   * Initiate a CSV writer with a ip, port and a logname which serves as the
```

```
     * filename
     *
     * @param logName
     * @param ip
     * @param port
     */
    public PerformanceLogger(String logName, InetAddress ip, int port) {
        String fileName = "PerformanceLog_" + logName + ".csv";
        sock = null;
        this.ip = ip;
        this.port = port;
        loadWriter(fileName);
        mbsc = ManagementFactory.getPlatformMBeanServer();
    }

    /**
     * Loads a writer, handles if the file doesn't exist or does and overwrites.
     *
     * @param filename
     */
    public void loadWriter(String filename) {
        try {
            writer = new PrintWriter(filename, "UTF-8");
        } catch (FileNotFoundException ex) {
            Logger.getLogger(PerformanceLogger.class.getName()).log(Level.SEVERE, null, ex);
        } catch (UnsupportedEncodingException ex) {
            Logger.getLogger(PerformanceLogger.class.getName()).log(Level.SEVERE, null, ex);
        }
        writeToFile("Tick Count, Memory Used %, Response Time, Requests, Logged Users, Thread Count, CPU %, activeRooms");
    }

    /**
     * Write a line to the CSV, this is usually done as: entry, entry, entry,
     * entry The line carrage is done automatically
     *
     * @param dataSet
     */
    public void writeToFile(String dataSet) {
        writer.println(dataSet);
    }

    /**
     * Write a system log to the specified writer file.
     *
     * @param tick
     * @param requestsInTick
     * @param loggedUsers
     */
    public void systemLog(int tick, int requestsInTick, int loggedUsers, int activeRooms) {
        OperatingSystemMXBean osMBean;
        double CPUloadpercent = 0;
        try {
            osMBean = ManagementFactory.newPlatformMXBeanProxy(
                    mbsc, ManagementFactory.OPERATING_SYSTEM_MXBEAN_NAME, OperatingSystemMXBean.class);
            CPUloadpercent = osMBean.getProcessCpuLoad() * 100;
        } catch (IOException ex) {
```

```
            Logger.getLogger(PerformanceLogger.class.getName()).log(Level.SEVERE, null, ex);
        }
        Runtime runtime = Runtime.getRuntime();
        long responseTime = responseTime();
        long maxMemory = runtime.maxMemory();
        long memoryHeld = runtime.totalMemory();
        float memoryUsedPercentage = (float) ((memoryHeld * 100) / maxMemory);
        String csvString = tick + ","
                + memoryUsedPercentage + ","
                + responseTime + ","
                + requestsInTick + ","
                + loggedUsers + ","
                + java.lang.Thread.activeCount() + ","
                 + CPUloadpercent + ","
                + activeRooms;
        writeToFile(csvString);
    }

    /**
     * Finds out the response time in ms.
     *
     * @return
     */
    public long responseTime() {
        initateResponseSocket(ip, port);
        long returnLong = 0;
        try {
            OutputStream os = sock.getOutputStream();
            InputStream is = sock.getInputStream();
            BufferedOutputStream bos = new BufferedOutputStream(os);
            OutputStreamWriter osw = new OutputStreamWriter(bos);
            String msg = "PING" + "\n";
            long startTime = System.currentTimeMillis();
            osw.write(msg);
            osw.flush();
            String responseString = readStream(is);
            if (responseString != null) {
                returnLong = System.currentTimeMillis() - startTime;
            } else {
                close();
            }
        } catch (IOException ex) {
            Logger.getLogger(PerformanceLogger.class.getName()).log(Level.SEVERE, null, ex);
        }

        return returnLong;
    }

    /**
     * Reads an entry from the socket, this is used to find out if there is a
     * response, the String it reads is irrelevant, but should get a response.
     *
     * @param is
     * @return
     */
    public String readStream(InputStream is) {
```

```
        InputStreamReader insr = new InputStreamReader(is);
        BufferedReader br = new BufferedReader(insr);
        String entireRequest = null;
        String nextLine = "";
        int timeOut = 0;
        try {
            while (timeOut < 1000) {
                if ((nextLine = br.readLine()) != null) {
                    entireRequest = entireRequest + nextLine;
                    if (!br.ready()) {
                        break;
                    } else {
                        entireRequest = entireRequest + "\n";
                    }
                } else {
                    timeOut++;
                    Thread.sleep(1);
                }
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return entireRequest;
    }

    /**
     * Closes the writer and saves the file, this must be called at the end of
     * the logging time else the file will be unreadable.
     */
    public void close() {
        writer.flush();
        writer.close();
    }

    /**
     * Starts up the socket, this is done every response time to find out if the
     * server goes down or not.
     *
     * @param ip
     * @param port
     */
    private void initateResponseSocket(InetAddress ip, int port) {
        try {
            sock = new Socket(ip, port);
        } catch (IOException ex) {
            Logger.getLogger(PerformanceLogger.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

ServerTerminal class:

```
package view;

import conclave.controllers.ServerController;
import java.util.List;
```

```
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.DefaultListModel;
import javax.swing.SwingWorker;

/**
 * This provides a basic view interaction with the server. Allows a superadmin
 * to add new admins, and start/stop the server.
 *
 * @author BradleyW
 */
public class ServerTerminal extends javax.swing.JFrame {

    private ServerController controller;
    DefaultListModel dlm1 = new DefaultListModel();
    DefaultListModel dlm2 = new DefaultListModel();

    public ServerTerminal() {
        this.setTitle("Conclave Server Terminal");
        initComponents();
        controller = ServerController.getInstance();
        keyUsers.setModel(dlm1);
        keys.setModel(dlm2);
        repopulateKeyList();
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        stopServer = new javax.swing.JButton();
        startServer = new javax.swing.JButton();
        jScrollPane1 = new javax.swing.JScrollPane();
        terminal = new javax.swing.JTextArea();
        adminName = new javax.swing.JTextField();
        jLabel1 = new javax.swing.JLabel();
        addAdmin = new javax.swing.JButton();
        jSeparator1 = new javax.swing.JSeparator();
        jLabel2 = new javax.swing.JLabel();
        jScrollPane2 = new javax.swing.JScrollPane();
        keys = new javax.swing.JList<>();
        generateNewKey = new javax.swing.JButton();
        viewKeyUsers = new javax.swing.JButton();
        jScrollPane3 = new javax.swing.JScrollPane();
        keyUsers = new javax.swing.JList<>();
        revokeKey = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        stopServer.setText("Stop Server");
        stopServer.addActionListener(new java.awt.event.ActionListener() {
```

```
    public void actionPerformed(java.awt.event.ActionEvent evt) {
      stopServerActionPerformed(evt);
    }
});

startServer.setText("Start Server");
startServer.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
      startServerActionPerformed(evt);
    }
});

terminal.setEditable(false);
terminal.setBackground(new java.awt.Color(220, 220, 220));
terminal.setColumns(20);
terminal.setRows(5);
terminal.setDisabledTextColor(new java.awt.Color(0, 0, 0));
jScrollPane1.setViewportView(terminal);

jLabel1.setText("Add an Admin:");

addAdmin.setText("Add");
addAdmin.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
      addAdminActionPerformed(evt);
    }
});

jLabel2.setText("Session Keys");

keys.setModel(new javax.swing.AbstractListModel<String>() {
    String[] strings = { "Item 1", "Item 2", "Item 3", "Item 4", "Item 5" };
    public int getSize() { return strings.length; }
    public String getElementAt(int i) { return strings[i]; }
});
jScrollPane2.setViewportView(keys);

generateNewKey.setText("Generate new Key");
generateNewKey.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
      generateNewKeyActionPerformed(evt);
    }
});

viewKeyUsers.setText("View Key Users");
viewKeyUsers.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
      viewKeyUsersActionPerformed(evt);
    }
});

keyUsers.setModel(new javax.swing.AbstractListModel<String>() {
    String[] strings = { "Item 1", "Item 2", "Item 3", "Item 4", "Item 5" };
    public int getSize() { return strings.length; }
    public String getElementAt(int i) { return strings[i]; }
});
```

```
    jScrollPane3.setViewportView(keyUsers);

    revokeKey.setText("Revoke Key");
    revokeKey.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        revokeKeyActionPerformed(evt);
      }
    });

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
      layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(layout.createSequentialGroup()
        .addContainerGap()
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addComponent(jScrollPane2, javax.swing.GroupLayout.Alignment.TRAILING)
          .addComponent(adminName)
          .addComponent(jScrollPane1)
          .addComponent(addAdmin, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
          .addGroup(layout.createSequentialGroup()
            .addComponent(jLabel2)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jSeparator1))
          .addComponent(jScrollPane3)
          .addGroup(layout.createSequentialGroup()
            .addComponent(jLabel1)
            .addGap(0, 0, Short.MAX_VALUE))
          .addGroup(layout.createSequentialGroup()
            .addComponent(generateNewKey)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(viewKeyUsers)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(revokeKey, javax.swing.GroupLayout.DEFAULT_SIZE, 119, Short.MAX_VALUE))
          .addGroup(layout.createSequentialGroup()
            .addComponent(startServer, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(stopServer)))
        .addContainerGap())
    );
    layout.setVerticalGroup(
      layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
        .addContainerGap()
        .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
          .addComponent(startServer)
          .addComponent(stopServer))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(jLabel1)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
```

```
            .addComponent(adminName, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(addAdmin)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
               .addComponent(jSeparator1, javax.swing.GroupLayout.PREFERRED_SIZE, 10, javax.swing.GroupLayout.PREFERRED_SIZE)
               .addComponent(jLabel2))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jScrollPane2, javax.swing.GroupLayout.PREFERRED_SIZE, 100, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
               .addComponent(generateNewKey)
               .addComponent(viewKeyUsers)
               .addComponent(revokeKey))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jScrollPane3, javax.swing.GroupLayout.DEFAULT_SIZE, 107, Short.MAX_VALUE)
            .addContainerGap())
      );

      pack();
   }// </editor-fold>

   private void stopServerActionPerformed(java.awt.event.ActionEvent evt) {
      terminal.setText("Shutdown begun, performance logger has saved. Terminal "
            + "\n will close soon.");
      controller.stopServer();
      SwingWorker closeWorker = new SwingWorker() {
         @Override
         protected Object doInBackground() throws Exception {
            controller.stopServer();
            try {
               Thread.sleep(5000);
               System.exit(-1);
            } catch (InterruptedException ex) {
               Logger.getLogger(ServerTerminal.class.getName()).log(Level.SEVERE, null, ex);
            }
            return null;
         }
      };
      closeWorker.execute();
   }

   private void startServerActionPerformed(java.awt.event.ActionEvent evt) {
      SwingWorker sw = new SwingWorker() {
         @Override
         protected Object doInBackground() throws Exception {
            controller.startServer();
            return null;
         }

         @Override
         protected void done() {
            terminal.setText("Server is online");
         }
      };
```

```
    sw.execute();
}

private void addAdminActionPerformed(java.awt.event.ActionEvent evt) {
    controller.addAdmin(adminName.getText());
    terminal.setText(adminName.getText() + " has been added as an admin.");
    adminName.setText("");
}

private void generateNewKeyActionPerformed(java.awt.event.ActionEvent evt) {
    String filename = controller.generateNewSessionKey();
    repopulateKeyList();
    terminal.setText("New Key has been generated, saved as: " + filename);
}

private void viewKeyUsersActionPerformed(java.awt.event.ActionEvent evt) {
    String keyValue = keys.getSelectedValue();
    if (keyValue != null) {
        repopulateUserListings();
        terminal.setText("Now displaying users of: " + keyValue.substring(0, 6) + "...");
    } else {
        terminal.setText("You must first select a key to view.");
    }
}

private void revokeKeyActionPerformed(java.awt.event.ActionEvent evt) {
    String keyValue = keys.getSelectedValue();
    if (keyValue != null) {
        controller.revokeKey(keyValue.substring(0, 32));
        terminal.setText("Key Revoked: " + keyValue);
        repopulateKeyList();
    } else {
        terminal.setText("You must first select a key to revoke.");
    }

}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
    /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
     * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
     */
    try {
        for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
            if ("Nimbus".equals(info.getName())) {
                javax.swing.UIManager.setLookAndFeel(info.getClassName());
                break;
            }
        }
    } catch (ClassNotFoundException ex) {
        java.util.logging.Logger.getLogger(ServerTerminal.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    } catch (InstantiationException ex) {
```

```
            java.util.logging.Logger.getLogger(ServerTerminal.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {
            java.util.logging.Logger.getLogger(ServerTerminal.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            java.util.logging.Logger.getLogger(ServerTerminal.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new ServerTerminal().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton addAdmin;
    private javax.swing.JTextField adminName;
    private javax.swing.JButton generateNewKey;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JScrollPane jScrollPane3;
    private javax.swing.JSeparator jSeparator1;
    private javax.swing.JList<String> keyUsers;
    private javax.swing.JList<String> keys;
    private javax.swing.JButton revokeKey;
    private javax.swing.JButton startServer;
    private javax.swing.JButton stopServer;
    private javax.swing.JTextArea terminal;
    private javax.swing.JButton viewKeyUsers;
    // End of variables declaration

    private void repopulateKeyList() {
        dlm2.clear();
        List<String> keyLists = controller.getAllKeys();
        try {
            if (!keyLists.isEmpty()) {
                for (String key : keyLists) {
                    dlm2.addElement(key);
                }
            }
        } catch (NullPointerException e) {

        }
    }

    private void repopulateUserListings() {
        dlm1.clear();
        String key = keys.getSelectedValue();
        try {
            List<String> userLists = controller.getKeyUsers(key.substring(0, 32));
            if (!userLists.isEmpty()) {
                for (String user : userLists) {
```

```
                dlm1.addElement(user);
            }
          }
      } catch (NullPointerException e) {

      }
    }
}
```

Main class:

```
package view;

/**
 *
 * @author BradleyW
 */
public class main {

    public static void main(String[] args) {
        ServerTerminal st = new ServerTerminal();
        st.setVisible(true);
    }
}
```

Client Program:

Main Class:

```
import conclaveclient.KeyInput;
import conclaveclient.LoginController;
import conclaveclient.LoginGUI;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
/**
 *
 * @author BradleyW
 */
public class main {

    public static void main(String[] args) throws IOException {
        boolean capturedKey = false;
        try {
            File secretKeyLocation = new File("secretkey.txt");
            if (secretKeyLocation != null) {
                BufferedReader br = new BufferedReader(new FileReader(secretKeyLocation));
                String line;
                while ((line = br.readLine()) != null) {
                    String secretKey = line.substring(0, 32);
                    String keyUser = line.substring(32, 37);
```

```
                capturedKey = true;
                LoginController controller = new LoginController(secretKey, keyUser);
                LoginGUI gui = new LoginGUI(controller);
                gui.setVisible(true);
            }
          }
        } catch (FileNotFoundException ex) {
        }
        if (!capturedKey) {
          System.out.println("Cannot load a 'secretkey.txt' file in Client directory");
          KeyInput ki = new KeyInput();
          ki.setVisible(true);
        }
    }
}
```

AdminPanel class:

```
package conclaveclient;

import conclaveinterfaces.IAdminInterface;
import conclaveinterfaces.IUserInterface;
import model.Message;
import java.rmi.RemoteException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author BradleyW
 */
public class AdminPanel extends javax.swing.JPanel {

    boolean adminController;
    IAdminInterface controller;

    /**
     * Creates new form AdminPanel
     */
    public AdminPanel(IUserInterface ui) {
        initComponents();
        adminController = true;
        controller = (IAdminInterface) ui;
        initilizeAdminContents();
        Thread adminUpdater = new Thread(new Runnable() {
            @Override
            public void run() {
                while (adminController) {
                    if (!AdminPanel.isShowing()) //Prevents contents being updated as used.
                    {
                        initilizeAdminContents();
                    }
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException ex) {
                        Logger.getLogger(AdminPanel.class.getName()).log(Level.SEVERE, null, ex);
                    }
```

```
        }

      }
    });
    adminUpdater.start();
}

/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    AdminPanel = new javax.swing.JPanel();
    jLabel10 = new javax.swing.JLabel();
    roomSelection = new javax.swing.JComboBox<>();
    roomOpenButton = new javax.swing.JButton();
    roomCloseButton = new javax.swing.JButton();
    jSeparator2 = new javax.swing.JSeparator();
    jLabel11 = new javax.swing.JLabel();
    roomName = new javax.swing.JTextField();
    jLabel12 = new javax.swing.JLabel();
    jLabel13 = new javax.swing.JLabel();
    roomTypeDropdown = new javax.swing.JComboBox<>();
    privateCheckbox = new javax.swing.JCheckBox();
    roomPassword = new javax.swing.JTextField();
    jLabel15 = new javax.swing.JLabel();
    jSeparator3 = new javax.swing.JSeparator();
    jLabel14 = new javax.swing.JLabel();
    playerSelection = new javax.swing.JComboBox<>();
    banCheckbox = new javax.swing.JCheckBox();
    banKickButton = new javax.swing.JButton();
    jLabel17 = new javax.swing.JLabel();
    oneWayMessageField = new javax.swing.JTextField();
    sendMessage = new javax.swing.JButton();
    createRoomButton = new javax.swing.JButton();
    jSeparator4 = new javax.swing.JSeparator();
    announcmementField = new javax.swing.JTextField();
    jLabel19 = new javax.swing.JLabel();
    postAnnouncementButton = new javax.swing.JButton();
    refresh = new javax.swing.JButton();
    jLabel1 = new javax.swing.JLabel();
    mute = new javax.swing.JButton();
    jSeparator1 = new javax.swing.JSeparator();

    jLabel10.setText("Rooms:");

    roomSelection.setModel(new javax.swing.DefaultComboBoxModel<>(new String[] { "Item 1", "Item 2", "Item 3", "Item 4" }));
    roomSelection.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        roomSelectionActionPerformed(evt);
      }
    });
```

```
roomOpenButton.setText("Open");
roomOpenButton.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    roomOpenButtonActionPerformed(evt);
  }
});

roomCloseButton.setText("Close");
roomCloseButton.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    roomCloseButtonActionPerformed(evt);
  }
});

jLabel11.setText("Rooms");

jLabel12.setText("Name:");

jLabel13.setText("Type:");

roomTypeDropdown.setModel(new javax.swing.DefaultComboBoxModel<>(new String[] { "Item 1", "Item 2", "Item 3", "Item 4" }));

privateCheckbox.setText("Private");
privateCheckbox.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    privateCheckboxActionPerformed(evt);
  }
});

roomPassword.setEnabled(false);

jLabel15.setText("Password:");

jLabel14.setText("Connections");

playerSelection.setModel(new javax.swing.DefaultComboBoxModel<>(new String[] { "Item 1", "Item 2", "Item 3", "Item 4" }));
playerSelection.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    playerSelectionActionPerformed(evt);
  }
});

banCheckbox.setText("Ban");

banKickButton.setText("Kick");
banKickButton.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    banKickButtonActionPerformed(evt);
  }
});

jLabel17.setText("Send One-Way Message:");

sendMessage.setText("Send");
sendMessage.addActionListener(new java.awt.event.ActionListener() {
```

```
    public void actionPerformed(java.awt.event.ActionEvent evt) {
      sendMessageActionPerformed(evt);
    }
  });

  createRoomButton.setText("Create");
  createRoomButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
      createRoomButtonActionPerformed(evt);
    }
  });

  jLabel19.setText("Server Announcements:");

  postAnnouncementButton.setText("Post");
  postAnnouncementButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
      postAnnouncementButtonActionPerformed(evt);
    }
  });

  refresh.setText("Refresh");
  refresh.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
      refreshActionPerformed(evt);
    }
  });

  jLabel1.setText("As entries will not update while editing, refresh using the button:");

  mute.setText("(Un)Mute");
  mute.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
      muteActionPerformed(evt);
    }
  });

  javax.swing.GroupLayout AdminPanelLayout = new javax.swing.GroupLayout(AdminPanel);
  AdminPanel.setLayout(AdminPanelLayout);
  AdminPanelLayout.setHorizontalGroup(
    AdminPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(AdminPanelLayout.createSequentialGroup()
      .addContainerGap()
      .addGroup(AdminPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(AdminPanelLayout.createSequentialGroup()
          .addComponent(jLabel11)
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
          .addComponent(jSeparator2))
        .addComponent(jSeparator1)
        .addGroup(AdminPanelLayout.createSequentialGroup()
          .addGroup(AdminPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(AdminPanelLayout.createSequentialGroup()
              .addComponent(jLabel14)
              .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
              .addComponent(jSeparator3))
            .addGroup(AdminPanelLayout.createSequentialGroup()
```

```
                .addGap(18, 18, 18)
                .addComponent(jLabel10)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(roomSelection, 0, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(roomOpenButton)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(roomCloseButton))
              .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, AdminPanelLayout.createSequentialGroup()
                .addComponent(jLabel19)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jSeparator4))
              .addGroup(AdminPanelLayout.createSequentialGroup()
                .addGap(7, 7, 7)
                .addComponent(playerSelection, javax.swing.GroupLayout.PREFERRED_SIZE, 379,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(banCheckbox)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(banKickButton, javax.swing.GroupLayout.DEFAULT_SIZE, 75, Short.MAX_VALUE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(mute))
              .addGroup(AdminPanelLayout.createSequentialGroup()
                .addComponent(announcmementField, javax.swing.GroupLayout.PREFERRED_SIZE, 467,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(postAnnouncementButton, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
              .addGroup(AdminPanelLayout.createSequentialGroup()
                .addGap(6, 6, 6)
                .addComponent(jLabel17)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(oneWayMessageField, javax.swing.GroupLayout.PREFERRED_SIZE, 320,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(sendMessage, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
              .addGroup(AdminPanelLayout.createSequentialGroup()
                .addGap(20, 20, 20)
                .addGroup(AdminPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                  .addGroup(AdminPanelLayout.createSequentialGroup()
                    .addComponent(jLabel12)
                    .addGap(18, 18, 18)
                    .addComponent(roomName))
                  .addGroup(AdminPanelLayout.createSequentialGroup()
                    .addComponent(jLabel13)
                    .addGap(24, 24, 24)
                    .addComponent(roomTypeDropdown, javax.swing.GroupLayout.PREFERRED_SIZE, 272,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGap(0, 0, Short.MAX_VALUE))
                  .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, AdminPanelLayout.createSequentialGroup()
                    .addGap(0, 0, Short.MAX_VALUE)
                    .addComponent(createRoomButton, javax.swing.GroupLayout.PREFERRED_SIZE, 126,
javax.swing.GroupLayout.PREFERRED_SIZE))))
              .addGroup(AdminPanelLayout.createSequentialGroup()
                .addGap(0, 0, Short.MAX_VALUE)
```

```
                    .addComponent(privateCheckbox)
                    .addGap(18, 18, 18)
                    .addComponent(jLabel15)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                    .addComponent(roomPassword, javax.swing.GroupLayout.PREFERRED_SIZE, 430,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                .addContainerGap())
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, AdminPanelLayout.createSequentialGroup()
                .addGap(0, 0, Short.MAX_VALUE)
                .addGroup(AdminPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, AdminPanelLayout.createSequentialGroup()
                        .addComponent(refresh, javax.swing.GroupLayout.PREFERRED_SIZE, 287, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(164, 164, 164))
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, AdminPanelLayout.createSequentialGroup()
                        .addComponent(jLabel1)
                        .addGap(123, 123, 123))))))
    );
    AdminPanelLayout.setVerticalGroup(
        AdminPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(AdminPanelLayout.createSequentialGroup()
            .addContainerGap()
            .addGroup(AdminPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                .addComponent(jSeparator2, javax.swing.GroupLayout.PREFERRED_SIZE, 10, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jLabel11))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(AdminPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(jLabel10, javax.swing.GroupLayout.Alignment.TRAILING)
                .addComponent(roomSelection, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGroup(AdminPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(roomOpenButton)
                    .addComponent(roomCloseButton)))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(AdminPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(roomName, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jLabel12))
            .addGroup(AdminPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(AdminPanelLayout.createSequentialGroup()
                    .addGroup(AdminPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addGroup(AdminPanelLayout.createSequentialGroup()
                            .addGap(4, 4, 4)
                            .addComponent(roomTypeDropdown, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addGroup(AdminPanelLayout.createSequentialGroup()
                            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                            .addComponent(jLabel13)))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addGroup(AdminPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(roomPassword, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(jLabel15)
                        .addComponent(privateCheckbox)))
                .addGroup(AdminPanelLayout.createSequentialGroup()
                    .addGap(4, 4, 4)
                    .addComponent(createRoomButton)))
```

```
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(AdminPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
              .addComponent(jSeparator3, javax.swing.GroupLayout.PREFERRED_SIZE, 10, javax.swing.GroupLayout.PREFERRED_SIZE)
              .addComponent(jLabel14))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(AdminPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
              .addComponent(playerSelection, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
              .addGroup(AdminPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(banCheckbox)
                .addComponent(banKickButton)
                .addComponent(mute)))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(AdminPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
              .addComponent(jLabel17)
              .addComponent(oneWayMessageField, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
              .addComponent(sendMessage))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(AdminPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
              .addComponent(jSeparator4, javax.swing.GroupLayout.PREFERRED_SIZE, 10, javax.swing.GroupLayout.PREFERRED_SIZE)
              .addComponent(jLabel19))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(AdminPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
              .addComponent(announcmementField)
              .addComponent(postAnnouncementButton, javax.swing.GroupLayout.PREFERRED_SIZE, 63,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addComponent(jSeparator1, javax.swing.GroupLayout.PREFERRED_SIZE, 10, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 16, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(refresh)
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);
        this.setLayout(layout);
        layout.setHorizontalGroup(
          layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addComponent(AdminPanel, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        );
        layout.setVerticalGroup(
          layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addGroup(layout.createSequentialGroup()
            .addComponent(AdminPanel, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(0, 0, Short.MAX_VALUE))
        );
    }// </editor-fold>

    private void roomSelectionActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }
```

```
private void roomOpenButtonActionPerformed(java.awt.event.ActionEvent evt) {
   if (adminController) {
      try {
         String roomselect = (String) roomSelection.getSelectedItem();
         controller.openRoom(roomselect);
         Message openRoomMsg = new Message("System", controller.getUsername(), roomselect + " is now opened", 2);
         controller.updateChatLog(openRoomMsg);
         initilizeAdminContents();
      } catch (RemoteException e) {

      }
   }
}

private void roomCloseButtonActionPerformed(java.awt.event.ActionEvent evt) {
   if (adminController) {
      try {
         String roomselect = (String) roomSelection.getSelectedItem();
         controller.closeRoom(roomselect);
         Message closeRoomMsg = new Message("System", controller.getUsername(), roomselect + " is now closed", 2);
         controller.updateChatLog(closeRoomMsg);
         initilizeAdminContents();
      } catch (RemoteException e) {
         e.printStackTrace();
      }
   }
}

private void privateCheckboxActionPerformed(java.awt.event.ActionEvent evt) {
   if (roomPassword.isEnabled()) {
      roomPassword.setEnabled(false);
   } else {
      roomPassword.setEnabled(true);
   }
}

private void playerSelectionActionPerformed(java.awt.event.ActionEvent evt) {
   // TODO add your handling code here:
}

private void banKickButtonActionPerformed(java.awt.event.ActionEvent evt) {
   if (adminController) {
      try {
         Message kickStatus = new Message("System", controller.getUsername(), "Cannot kick/ban that user", 2);
         String connectionSelectUsername = (String) playerSelection.getSelectedItem();
         boolean banned = banCheckbox.isSelected();
         controller.kickUser(connectionSelectUsername, banned);
         kickStatus.setMsgText("That user has been kicked/banned");
         controller.updateChatLog(kickStatus);
         initilizeAdminContents();
      } catch (RemoteException e) {
      }
   }
}

private void sendMessageActionPerformed(java.awt.event.ActionEvent evt) {
```

```
    String msg = oneWayMessageField.getText();
    String username = (String) playerSelection.getSelectedItem();
    Message sendMessage = new Message(username, username, msg, 4);
    if (msg != null) {
      try {
        controller.sendAdminMessage(sendMessage, username);
        controller.updateChatLog(sendMessage);
      } catch (RemoteException ex) {
        Logger.getLogger(SwingGUI.class.getName()).log(Level.SEVERE, null, ex);
      }
    }
  }

  private void createRoomButtonActionPerformed(java.awt.event.ActionEvent evt) {
    if (adminController) {
      try {
        Message createRoomStatus = new Message("System", controller.getUsername(), "Room creation has failed", 2);
        String newRoomName = roomName.getText();
        String newRoomTypeName = (String) roomTypeDropdown.getSelectedItem();
        int roomType = 0;
        switch (newRoomTypeName) {
          case "TextRoom":
            roomType = 1;
            break;
          case "ConferenceRoom":
            roomType = 2;
            break;
        }
        if (privateCheckbox.isSelected()) {
          String roomPasswordptext = roomPassword.getText();
          if (roomPasswordptext != null) {
            controller.addRoom(newRoomName, roomType, roomPasswordptext);
            createRoomStatus.setMsgText("Private room successfully created");
          } else {
            createRoomStatus.setMsgText("You must enter a password to create a private room");
          }
        } else {
          controller.addRoom(newRoomName, roomType);
          createRoomStatus.setMsgText("Open room successfully created");
        }
        controller.updateChatLog(createRoomStatus);
        initilizeAdminContents();
      } catch (RemoteException e) {

      }
    }
  }

  private void postAnnouncementButtonActionPerformed(java.awt.event.ActionEvent evt) {
    if (adminController) {
      try {
        String msg = announcmementField.getText();
        if (msg != null) {
          controller.postAnnouncment(msg);
          controller.updateChatLog(new Message("System", controller.getUsername(), "You have posted a new announcement!", 2));
          announcmementField.setText("");
```

```
            }
        } catch (RemoteException e) {

        }
    }
}

private void refreshActionPerformed(java.awt.event.ActionEvent evt) {
    initilizeAdminContents();
}

private void muteActionPerformed(java.awt.event.ActionEvent evt) {
    if (adminController) {
        try {
            Message muteStatus = new Message("System", controller.getUsername(), "Cannot mute that user", 2);
            String connectionSelectUsername = (String) playerSelection.getSelectedItem();
            if (!controller.isMuted(connectionSelectUsername))
            {
                controller.censorUser(connectionSelectUsername);
                muteStatus.setMsgText("User: " + connectionSelectUsername + " has been muted");
            } else {
                controller.uncensorUser(connectionSelectUsername);
                muteStatus.setMsgText("User: " + connectionSelectUsername + " has been unmuted");
            }
            controller.updateChatLog(muteStatus);
        } catch (RemoteException ex) {
            Logger.getLogger(AdminPanel.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}

private void initilizeAdminContents() {
    roomSelection.removeAllItems();
    roomTypeDropdown.removeAllItems();
    playerSelection.removeAllItems();
    adminController = true;
    try {
        List<String> roomNames = controller.getRoomNames();
        for (String name : roomNames) {
            roomSelection.addItem(name);
        }
        List<String> supportedRoomTypes = controller.getSupportedRoomTypes();
        for (String roomType : supportedRoomTypes) {
            roomTypeDropdown.addItem(roomType);
        }
        List<String> activeUserNames = controller.getAllConnectedUsernames();
        for (String username : activeUserNames) {
            playerSelection.addItem(username);
        }
    } catch (RemoteException e) {

    }
}

// Variables declaration - do not modify
private javax.swing.JPanel AdminPanel;
```

```
    private javax.swing.JTextField announcmementField;
    private javax.swing.JCheckBox banCheckbox;
    private javax.swing.JButton banKickButton;
    private javax.swing.JButton createRoomButton;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel12;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel14;
    private javax.swing.JLabel jLabel15;
    private javax.swing.JLabel jLabel17;
    private javax.swing.JLabel jLabel19;
    private javax.swing.JSeparator jSeparator1;
    private javax.swing.JSeparator jSeparator2;
    private javax.swing.JSeparator jSeparator3;
    private javax.swing.JSeparator jSeparator4;
    private javax.swing.JButton mute;
    private javax.swing.JTextField oneWayMessageField;
    private javax.swing.JComboBox<String> playerSelection;
    private javax.swing.JButton postAnnouncementButton;
    private javax.swing.JCheckBox privateCheckbox;
    private javax.swing.JButton refresh;
    private javax.swing.JButton roomCloseButton;
    private javax.swing.JTextField roomName;
    private javax.swing.JButton roomOpenButton;
    private javax.swing.JTextField roomPassword;
    private javax.swing.JComboBox<String> roomSelection;
    private javax.swing.JComboBox<String> roomTypeDropdown;
    private javax.swing.JButton sendMessage;
    // End of variables declaration
}
```

KeyInput class:

```
package conclaveclient;
import conclaveclient.LoginController;
import conclaveclient.LoginGUI;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
/**
 *
 * @author BradleyW
 */
public class KeyInput extends javax.swing.JFrame {

    /**
     * Creates new form KeyInput
     */
    public KeyInput() {
        initComponents();
        this.setTitle("Conclave Key-Not-Found Window");
    }

    /**
```

```
    * This method is called from within the constructor to initialize the form.
    * WARNING: Do NOT modify this code. The content of this method is always
    * regenerated by the Form Editor.
    */
   @SuppressWarnings("unchecked")
   // <editor-fold defaultstate="collapsed" desc="Generated Code">
   private void initComponents() {

      jScrollPane1 = new javax.swing.JScrollPane();
      output = new javax.swing.JTextArea();
      keyinput = new javax.swing.JTextField();
      submitKey = new javax.swing.JButton();
      jLabel1 = new javax.swing.JLabel();

      setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

      output.setEditable(false);
      output.setColumns(20);
      output.setRows(5);
      output.setText("The Conclave key could not be found in the ConclaveClient directory, please make sure that the\nkey is inside the
directory, named \"secretkey\" with a filetype of txt. Once this has been corrected,\nreload the client by closing this window and launching
again. If this file is not provided, please\nmanually enter the key in the below input and submit.");
      jScrollPane1.setViewportView(output);

      submitKey.setText("Submit");
      submitKey.addActionListener(new java.awt.event.ActionListener() {
         public void actionPerformed(java.awt.event.ActionEvent evt) {
            submitKeyActionPerformed(evt);
         }
      });

      jLabel1.setText("Enter your user key below:");

      javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
      getContentPane().setLayout(layout);
      layout.setHorizontalGroup(
         layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
         .addGroup(layout.createSequentialGroup()
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
               .addGroup(layout.createSequentialGroup()
                  .addContainerGap()
                  .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                     .addComponent(keyinput)
                     .addGroup(layout.createSequentialGroup()
                        .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 569, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(0, 0, Short.MAX_VALUE))))
               .addGroup(layout.createSequentialGroup()
                  .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                     .addGroup(layout.createSequentialGroup()
                        .addGap(182, 182, 182)
                        .addComponent(submitKey, javax.swing.GroupLayout.PREFERRED_SIZE, 173, javax.swing.GroupLayout.PREFERRED_SIZE))
                     .addGroup(layout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(jLabel1)))
                  .addGap(0, 0, Short.MAX_VALUE)))
            .addContainerGap())
```

```
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGap(31, 31, 31)
                .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 86, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(18, 18, 18)
                .addComponent(jLabel1)
                .addGap(2, 2, 2)
                .addComponent(keyinput, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(submitKey)
                .addContainerGap(23, Short.MAX_VALUE))
        );

        pack();
    }// </editor-fold>

    private void submitKeyActionPerformed(java.awt.event.ActionEvent evt) {
        String wholeString = keyinput.getText();
        if (wholeString.length() == 37) {
            String secKey = wholeString.substring(0, 32);
            String secKeyUser = wholeString.substring(32, 37);
            initiateLoginWindow(secKey, secKeyUser);
            try {
                FileOutputStream fos = new FileOutputStream(new File("secretkey.txt"));
                fos.write(wholeString.getBytes());
                fos.flush();
                fos.close();
            } catch (IOException ex) {
                Logger.getLogger(KeyInput.class.getName()).log(Level.SEVERE, null, ex);
            }
        } else {
            output.setText("Invalid format of secret key.");
        }
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {
            java.util.logging.Logger.getLogger(KeyInput.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
```

```
        } catch (InstantiationException ex) {
            java.util.logging.Logger.getLogger(KeyInput.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {
            java.util.logging.Logger.getLogger(KeyInput.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            java.util.logging.Logger.getLogger(KeyInput.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new KeyInput().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JLabel jLabel1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextField keyinput;
    private javax.swing.JTextArea output;
    private javax.swing.JButton submitKey;
    // End of variables declaration

    public void initiateLoginWindow(String secKey, String secKeyUser) {
        this.setVisible(false);
        LoginController controller;
        controller = new LoginController(secKey, secKeyUser);
        LoginGUI gui = new LoginGUI(controller);
        gui.setVisible(true);
    }

}
```

LoginController class:

```
package conclaveclient;

import conclaveclient.Handlers.utlity.PacketUtil;
import static conclaveclient.rmi.RMISecurityPolicyLoader.LoadDefaultPolicy;
import conclaveinterfaces.IUserInterface;
import java.io.IOException;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.IllegalFormatException;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author BradleyW
 */
public class LoginController {
```

```
private boolean connected;
private PacketUtil packetUtil;
private static String seckey;
private static String seckeyUser;

InetAddress ip;
int port;

public LoginController(String skey, String skeyUser) {
   connected = false;
   LoadDefaultPolicy();

   this.seckey = skey;
   this.seckeyUser = skeyUser;
}

public String connect(String iip, String iport) {
   String returnString = "";
   connected = false;
   if (!connected) {
      try {
         if (!iport.matches("[0-9]+") || !iip.matches("[0-9]+") && iip.length() <= 4) {
            returnString = "Bad format of IP/Port";
         } else {
            ip = InetAddress.getByName(iip);
            port = Integer.parseInt(iport);
            packetUtil = new PacketUtil(ip, port, seckey, seckeyUser);
            packetUtil.sendPacketRequest("PING");
            returnString = packetUtil.readStream();
            if (returnString.contains("100")) {
               connected = true;
               returnString = "Connected to server at: " + ip.toString() + ":" + port;
            } else if (returnString!=null)
            {
               //we do not change the returnstring. This will usually be a respnose
               //such as invalid key.
            } else {
               packetUtil = null;
               returnString = "Failure to connect";
            }
         }
      } catch (UnknownHostException e) {
         returnString = "Cannot find a conclave server on that IP or Port";
      } catch (IOException e) {
         returnString = "Cannot find a conclave server on that IP or Port";
      } catch (IllegalFormatException e) {
         returnString = "Bad format of IP/Port";
      }
   } else {
      returnString = "You are already connected!";
   }
   if (packetUtil != null) {
      packetUtil.close();
      packetUtil = null;
   }
```

```
    return returnString;
  }

  public String login(String username, String password) throws RemoteException {
    String returnString = "Error initiating login";
    if (connected) {
      try {
        packetUtil = new PacketUtil(ip, port, seckey, seckeyUser);
        packetUtil.sendPacketRequest("LOGIN " + username + " " + password);
        String responseString = packetUtil.readStream();
        String[] commandWords = responseString.split("//s+");
        if (commandWords[0].contains("100")) {
          Registry registry = LocateRegistry.getRegistry(ip.getHostAddress(), 9807);
          IUserInterface ui = (IUserInterface) registry.lookup(username);
          SwingGUI newSwingGui = new SwingGUI(ui);
          newSwingGui.setVisible(true);
        }
        returnString = responseString;
      } catch (UnknownHostException e) {
        returnString = "That conclave server is currently down";
      } catch (NotBoundException e) {
        returnString = "Cannot connect to server registry";
      } catch (IOException ex) {
        returnString = "IOException: " + ex.getLocalizedMessage();
        Logger.getLogger(LoginController.class.getName()).log(Level.SEVERE, null, ex);
      }
    } else {
      returnString = "You must first connect to a conclave server";
    }
    if (packetUtil != null) {
      packetUtil.close();
      packetUtil = null;
    }
    return returnString;
  }

  public boolean isConnected() {
    return connected;
  }

  public String createAccount(String username, String password) {
    String returnString = "";
    if (connected) {
      try {
        packetUtil = new PacketUtil(ip, port, seckey, seckeyUser);
        packetUtil.sendPacketRequest("SETUP-ACCOUNT " + username + " " + password);
        returnString = packetUtil.readStream();
      } catch (UnknownHostException e) {
        returnString = "That conclave server is currently down";
      } catch (IOException ex) {
        Logger.getLogger(LoginController.class.getName()).log(Level.SEVERE, null, ex);
      }
    } else {
      returnString = "You must first connect to a conclave server";
    }
    if (packetUtil != null) {
```

```
            packetUtil.close();
            packetUtil = null;
        }
        return returnString;
    }
}
```

LoginGUI class:

```
package conclaveclient;

import java.rmi.RemoteException;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.SwingWorker;

/**
 *
 * @author BradleyW
 */
public class LoginGUI extends javax.swing.JFrame {

    private static LoginController loginController;

    public LoginGUI(LoginController controller) {
        this.setTitle("Conclave Login");
        loginController = controller;
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jMenu1 = new javax.swing.JMenu();
        jScrollPane2 = new javax.swing.JScrollPane();
        jTable1 = new javax.swing.JTable();
        jScrollPane3 = new javax.swing.JScrollPane();
        jTextArea1 = new javax.swing.JTextArea();
        jMenuItem1 = new javax.swing.JMenuItem();
        usernameForm = new javax.swing.JTextField();
        usernameLabel = new javax.swing.JLabel();
        passwordForm = new javax.swing.JTextField();
        passwordLabel = new javax.swing.JLabel();
        loginLabel2 = new javax.swing.JLabel();
        loginButton = new javax.swing.JButton();
        createAccountButton = new javax.swing.JButton();
        jSeparator3 = new javax.swing.JSeparator();
        ipForm = new javax.swing.JTextField();
        portForm = new javax.swing.JTextField();
```

```
jScrollPane1 = new javax.swing.JScrollPane();
statusOutput = new javax.swing.JTextArea();
ipLabel = new javax.swing.JLabel();
portLabel = new javax.swing.JLabel();
connectButton = new javax.swing.JButton();
jScrollPane4 = new javax.swing.JScrollPane();
loginOutput = new javax.swing.JTextArea();
jSeparator4 = new javax.swing.JSeparator();
jLabel1 = new javax.swing.JLabel();
jLabel2 = new javax.swing.JLabel();
jLabel3 = new javax.swing.JLabel();

jMenu1.setText("jMenu1");

jTable1.setModel(new javax.swing.table.DefaultTableModel(
  new Object [][] {
    {null, null, null, null},
    {null, null, null, null},
    {null, null, null, null},
    {null, null, null, null}
  },
  new String [] {
    "Title 1", "Title 2", "Title 3", "Title 4"
  }
));
jScrollPane2.setViewportView(jTable1);

jTextArea1.setColumns(20);
jTextArea1.setRows(5);
jScrollPane3.setViewportView(jTextArea1);

jMenuItem1.setText("jMenuItem1");

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

usernameForm.setNextFocusableComponent(passwordForm);
usernameForm.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    usernameFormActionPerformed(evt);
  }
});

usernameLabel.setText("Username:");

passwordLabel.setText("Password:");

loginLabel2.setFont(new java.awt.Font("Tahoma", 2, 11)); // NOI18N
loginLabel2.setText("Login:");

loginButton.setText("Login");
loginButton.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    loginButtonActionPerformed(evt);
  }
});
```

```
        createAccountButton.setText("Create New Account");
        createAccountButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                createAccountButtonActionPerformed(evt);
            }
        });

        ipForm.setText("192.168.0.3");
        ipForm.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                ipFormActionPerformed(evt);
            }
        });

        portForm.setText("20003");
        portForm.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                portFormActionPerformed(evt);
            }
        });

        jScrollPane1.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
        jScrollPane1.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL_SCROLLBAR_NEVER);

        statusOutput.setBackground(new java.awt.Color(230, 230, 230));
        statusOutput.setColumns(20);
        statusOutput.setFont(statusOutput.getFont().deriveFont(statusOutput.getFont().getStyle() | java.awt.Font.BOLD,
statusOutput.getFont().getSize()+2));
        statusOutput.setForeground(new java.awt.Color(0, 51, 255));
        statusOutput.setRows(5);
        statusOutput.setDisabledTextColor(new java.awt.Color(0, 0, 0));
        statusOutput.setEnabled(false);
        jScrollPane1.setViewportView(statusOutput);

        ipLabel.setText("Server IP:");

        portLabel.setText("Port:");

        connectButton.setText("Connect");
        connectButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                connectButtonActionPerformed(evt);
            }
        });

        jScrollPane4.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
        jScrollPane4.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL_SCROLLBAR_NEVER);

        loginOutput.setEditable(false);
        loginOutput.setBackground(new java.awt.Color(230, 230, 230));
        loginOutput.setColumns(20);
        loginOutput.setFont(loginOutput.getFont().deriveFont(loginOutput.getFont().getStyle() | java.awt.Font.BOLD,
loginOutput.getFont().getSize()+3));
        loginOutput.setRows(5);
        jScrollPane4.setViewportView(loginOutput);
```

```
    jLabel1.setFont(new java.awt.Font("Tahoma", 2, 11)); // NOI18N
    jLabel1.setText("Connect:");

    jLabel2.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
    jLabel2.setText("Conclave. V1.1");

    jLabel3.setIcon(new javax.swing.ImageIcon(getClass().getResource("/conclaveclient/image/ConclaveLogo.png"))); // NOI18N

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
      layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(layout.createSequentialGroup()
        .addContainerGap()
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
          .addComponent(jScrollPane4, javax.swing.GroupLayout.Alignment.LEADING)
          .addGroup(layout.createSequentialGroup()
            .addGap(0, 0, Short.MAX_VALUE)
            .addComponent(loginLabel2)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jSeparator3, javax.swing.GroupLayout.PREFERRED_SIZE, 381, javax.swing.GroupLayout.PREFERRED_SIZE))
          .addGroup(javax.swing.GroupLayout.Alignment.LEADING, layout.createSequentialGroup()
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
              .addGroup(javax.swing.GroupLayout.Alignment.LEADING, layout.createSequentialGroup()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                  .addGroup(layout.createSequentialGroup()
                    .addComponent(usernameLabel)
                    .addGap(14, 14, 14))
                  .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
                    .addComponent(passwordLabel)
                    .addGap(18, 18, 18)))
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
                  .addGroup(layout.createSequentialGroup()
                    .addComponent(usernameForm, javax.swing.GroupLayout.PREFERRED_SIZE, 188,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(loginButton, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
                  .addGroup(layout.createSequentialGroup()
                    .addComponent(passwordForm, javax.swing.GroupLayout.PREFERRED_SIZE, 188,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(createAccountButton))))
              .addComponent(jScrollPane1, javax.swing.GroupLayout.Alignment.LEADING, javax.swing.GroupLayout.PREFERRED_SIZE,
416, javax.swing.GroupLayout.PREFERRED_SIZE)
              .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING, false)
                .addGroup(javax.swing.GroupLayout.Alignment.LEADING, layout.createSequentialGroup()
                  .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
                    .addGroup(layout.createSequentialGroup()
                      .addComponent(portLabel, javax.swing.GroupLayout.PREFERRED_SIZE, 86,
javax.swing.GroupLayout.PREFERRED_SIZE)
                      .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                      .addComponent(portForm))
                    .addGroup(layout.createSequentialGroup()
                      .addComponent(ipLabel, javax.swing.GroupLayout.PREFERRED_SIZE, 86, javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(ipForm, javax.swing.GroupLayout.PREFERRED_SIZE, 166,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(connectButton, javax.swing.GroupLayout.PREFERRED_SIZE, 152,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGroup(javax.swing.GroupLayout.Alignment.LEADING, layout.createSequentialGroup()
                    .addComponent(jLabel1)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(jLabel3)
                        .addComponent(jSeparator4))))
            .addGroup(javax.swing.GroupLayout.Alignment.LEADING, layout.createSequentialGroup()
                .addGap(145, 145, 145)
                .addComponent(jLabel2)))
        .addGap(0, 0, Short.MAX_VALUE)))
    .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
);
layout.setVerticalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
        .addGap(19, 19, 19)
        .addComponent(jLabel3)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 32, Short.MAX_VALUE)
        .addComponent(jLabel2)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jSeparator4, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.PREFERRED_SIZE, 10,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(jLabel1, javax.swing.GroupLayout.Alignment.TRAILING))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(ipForm, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(ipLabel))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(portForm, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(portLabel)))
            .addComponent(connectButton, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.PREFERRED_SIZE, 49,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 35, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jSeparator3, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.PREFERRED_SIZE, 10,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(loginLabel2, javax.swing.GroupLayout.Alignment.TRAILING))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(usernameLabel)
            .addComponent(usernameForm, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
                .addComponent(loginButton))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(passwordLabel)
                .addComponent(passwordForm, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(createAccountButton))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jScrollPane4, javax.swing.GroupLayout.PREFERRED_SIZE, 42, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addContainerGap())
        );

        pack();
    }// </editor-fold>

    private void loginButtonActionPerformed(java.awt.event.ActionEvent evt) {
        final LoginGUI window = this;
        SwingWorker sw = new SwingWorker() {
            String loginStatus = "ERROR WHILST LOGGING IN";
            @Override
            protected Object doInBackground() throws Exception {
                try {
                    if (loginController.isConnected()) {
                        String username = usernameForm.getText();
                        String password = passwordForm.getText();
                        if (username != null && password != null) {
                            if (username.length() < 5) {
                                loginStatus = "Username too short.";
                            } else if (username.length() < 5) {
                                loginStatus = "Password too short.";
                            } else {
                                setLoginStatusText("Logging in..");
                                String responseString = loginController.login(username, password);
                                if (responseString.contains("100")) {
                                    loginStatus = "Successful login";
                                    window.setVisible(false);
                                } else {
                                    loginStatus = responseString;
                                }
                            }
                        } else {
                            loginStatus = "You must enter all the fields";
                        }
                    } else {
                        loginStatus = "You must first connect to a Conclave server";
                    }
                } catch (NumberFormatException e) {
                    loginStatus = "You must enter all the fields appropriatly.";
                } catch (RemoteException ex) {
                    loginStatus = "Error connecting to specified server.";
                    Logger.getLogger(LoginGUI.class.getName()).log(Level.SEVERE, null, ex);
                }
                return null;
            }

            @Override
```

```
        protected void done() {
            setLoginStatusText(loginStatus);
        }
    };
    sw.execute();
    try {
        sw.get(5, TimeUnit.SECONDS);
    } catch (InterruptedException ex) {
        setLoginStatusText("Request timed-out");
    } catch (ExecutionException ex) {
        Logger.getLogger(LoginGUI.class.getName()).log(Level.SEVERE, null, ex);
    } catch (TimeoutException ex) {
        setLoginStatusText("Request timed-out");
    }
}

private void createAccountButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String creationStatus = "ERROR WHILST CREATING ACCOUNT";
    try {
        if (loginController.isConnected()) {
            String username = usernameForm.getText();
            String password = passwordForm.getText();
            if (username != null && password != null) {
                setLoginStatusText("Creating account..");
                creationStatus = loginController.createAccount(username, password);
            } else {
                creationStatus = "You must enter all the fields";
            }
        } else {
            creationStatus = "You must first connect to a Conclave server";
        }
    } catch (NumberFormatException e) {
        creationStatus = "You must enter all the fields appropriatly.";
    }
    setLoginStatusText(creationStatus);
}

private void ipFormActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

private void portFormActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

private void connectButtonActionPerformed(java.awt.event.ActionEvent evt) {

    final String IP = ipForm.getText();
    final String port = portForm.getText();
    SwingWorker sw = new SwingWorker() {
        String connectionStatus = "Cannot find an online server at that IP/Port";

        @Override
        protected Object doInBackground() throws Exception {
            if (IP.length() > 6 && port.length() > 1) {
                setConnectionStatusText("Connecting to server...");
```

```
            String response = loginController.connect(IP, port);
            System.out.println(response);
            if (loginController.isConnected()) {
               connectionStatus = response;
            } else {
               connectionStatus = "Cannot connect to this server.";
            }
            connectionStatus = response;
         } else {
            connectionStatus = "You must enter all the fields appropriatly.";
         }
         return null;
      }

      @Override
      protected void done() {
         setConnectionStatusText(connectionStatus);
      }
   };
   sw.execute();
}

private void usernameFormActionPerformed(java.awt.event.ActionEvent evt) {
   // TODO add your handling code here:
}

private void setLoginStatusText(String text) {
   loginOutput.setText("");
   loginOutput.setText(text);
}

private void setConnectionStatusText(String text) {
   statusOutput.setText("");
   statusOutput.setText(text);
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
   /* Set the Nimbus look and feel */
   //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
   /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
    * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
    */
   try {
      for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
         if ("Nimbus".equals(info.getName())) {
            javax.swing.UIManager.setLookAndFeel(info.getClassName());
            break;
         }
      }
   } catch (ClassNotFoundException ex) {
      java.util.logging.Logger.getLogger(LoginGUI.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
   } catch (InstantiationException ex) {
      java.util.logging.Logger.getLogger(LoginGUI.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
```

```
        } catch (IllegalAccessException ex) {
            java.util.logging.Logger.getLogger(LoginGUI.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            java.util.logging.Logger.getLogger(LoginGUI.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new LoginGUI(loginController).setVisible(true);
            }
        });

    }

    // Variables declaration - do not modify
    private javax.swing.JButton connectButton;
    private javax.swing.JButton createAccountButton;
    private javax.swing.JTextField ipForm;
    private javax.swing.JLabel ipLabel;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JMenu jMenu1;
    private javax.swing.JMenuItem jMenuItem1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JScrollPane jScrollPane3;
    private javax.swing.JScrollPane jScrollPane4;
    private javax.swing.JSeparator jSeparator3;
    private javax.swing.JSeparator jSeparator4;
    private javax.swing.JTable jTable1;
    private javax.swing.JTextArea jTextArea1;
    private javax.swing.JButton loginButton;
    private javax.swing.JLabel loginLabel2;
    private javax.swing.JTextArea loginOutput;
    private javax.swing.JTextField passwordForm;
    private javax.swing.JLabel passwordLabel;
    private javax.swing.JTextField portForm;
    private javax.swing.JLabel portLabel;
    private javax.swing.JTextArea statusOutput;
    private javax.swing.JTextField usernameForm;
    private javax.swing.JLabel usernameLabel;
    // End of variables declaration
}
```

SwingGUI class:

```
package conclaveclient;

import model.Announcement;
import model.ConnectionsLog;
import model.ConnectionEntry;
import model.Message;
import conclaveclient.Conference.display.BroadcastPanel;
import conclaveinterfaces.IUserInterface;
import java.awt.Dimension;
```

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.rmi.RemoteException;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JSeparator;
import javax.swing.SwingConstants;

/**
 *
 * @author BradleyW
 */
public class SwingGUI extends javax.swing.JFrame {

    public IUserInterface client;
    private boolean inRoom;
    private int lastMessageLine;
    private ArrayList<String> chatlogViewCategories = new ArrayList<String>();
    private BroadcastPanel broadcastPanel;
    private static int lastExport;

    public SwingGUI(IUserInterface ui) {
        try {
            ui.connect(); //Ensure the UI does not get thrown from the server.
            this.setTitle("Conclave");
            lastExport = 0;
            initComponents();
            client = ui;
            inRoom = false;
            welcomeLabel.setText("Welcome to Conclave, " + ui.getUsername());
            setConnectionsArea(client.viewAllConnections());
            buildFrontpage();
            setVisible(true);
            chatlogViewCategories.add("Room");
            chatlogViewCategories.add("System");
            chatlogViewCategories.add("Private");
            chatlogViewCategories.add("Admin");
            if (client.getType() == 1) {
                updateChatlog(new Message("Conclave", "Conclave", "You have logged in as a User", 2));
            } else if (client.getType() == 2) {
                updateChatlog(new Message("Conclave", "Conclave", "You have logged in as an Admin", 2));
                initilizeAdminTab();
            }
            startUpdates();
            pack();
            Runtime.getRuntime().addShutdownHook(new Thread() {
                @Override
```

```
            public void run() {
                try {
                    System.out.println("DISCONNECTING");
                    client.leaveServer();
                } catch (RemoteException ex) {
                    Logger.getLogger(SwingGUI.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        });
    } catch (RemoteException e) {
        e.printStackTrace();
    }
}

/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    jMenuItem1 = new javax.swing.JMenuItem();
    jMenuBar1 = new javax.swing.JMenuBar();
    jMenu1 = new javax.swing.JMenu();
    jMenu2 = new javax.swing.JMenu();
    jLabel1 = new javax.swing.JLabel();
    passwordFormPanel = new javax.swing.JPanel();
    roomPasswordEntry = new javax.swing.JTextField();
    roomPasswordSubmit = new javax.swing.JButton();
    jLabel2 = new javax.swing.JLabel();
    jLabel3 = new javax.swing.JLabel();
    jSeparator1 = new javax.swing.JSeparator();
    privateMsgPanel = new javax.swing.JPanel();
    pmField = new javax.swing.JTextField();
    pmSendButton = new java.awt.Button();
    pmLabel = new javax.swing.JLabel();
    filtersMsg = new javax.swing.ButtonGroup();
    frontpage = new javax.swing.JPanel();
    announcments = new java.awt.List();
    jScrollPane1 = new javax.swing.JScrollPane();
    connectionsScrollPanel = new javax.swing.JScrollPane();
    connectionsPanel = new javax.swing.JPanel();
    textLog = new java.awt.TextArea();
    sendMessageButton = new javax.swing.JButton();
    welcomeLabel = new javax.swing.JLabel();
    refreshConnections = new javax.swing.JButton();
    disconnectButton = new javax.swing.JButton();
    leaveRoomButton = new javax.swing.JButton();
    jScrollPane3 = new javax.swing.JScrollPane();
    textInputArea = new javax.swing.JTextArea();
    exportChatLogButton = new javax.swing.JButton();
    chatlogFilterPanel = new javax.swing.JPanel();
    roomFilterCheckbox = new javax.swing.JCheckBox();
    systemFilterCheckbox = new javax.swing.JCheckBox();
```

```
    privateFilterCheckbox = new javax.swing.JCheckBox();
    adminFilterCheckbox = new javax.swing.JCheckBox();
    filterLabel = new javax.swing.JLabel();
    tabbedPanel = new javax.swing.JTabbedPane();
    interactablePanel = new javax.swing.JPanel();

    jMenuItem1.setText("jMenuItem1");

    jMenu1.setText("File");
    jMenuBar1.add(jMenu1);

    jMenu2.setText("Edit");
    jMenuBar1.add(jMenu2);

    jLabel1.setText("jLabel1");

    roomPasswordEntry.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        roomPasswordEntryActionPerformed(evt);
      }
    });

    roomPasswordSubmit.setText("Submit");
    roomPasswordSubmit.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        roomPasswordSubmitActionPerformed(evt);
      }
    });

    jLabel2.setText("Password:");

    jLabel3.setText("This is a private room.");

    javax.swing.GroupLayout passwordFormPanelLayout = new javax.swing.GroupLayout(passwordFormPanel);
    passwordFormPanel.setLayout(passwordFormPanelLayout);
    passwordFormPanelLayout.setHorizontalGroup(
      passwordFormPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, passwordFormPanelLayout.createSequentialGroup()
        .addContainerGap(31, Short.MAX_VALUE)
        .addGroup(passwordFormPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addComponent(jLabel3)
          .addGroup(passwordFormPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(passwordFormPanelLayout.createSequentialGroup()
              .addGap(16, 16, 16)
              .addComponent(roomPasswordSubmit)
              .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 24, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
passwordFormPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
              .addComponent(jLabel2)
              .addComponent(roomPasswordEntry, javax.swing.GroupLayout.PREFERRED_SIZE, 113,
javax.swing.GroupLayout.PREFERRED_SIZE))))
        .addGap(25, 25, 25))
      .addComponent(jSeparator1)
    );
    passwordFormPanelLayout.setVerticalGroup(
      passwordFormPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, passwordFormPanelLayout.createSequentialGroup()
          .addContainerGap()
          .addComponent(jLabel3, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
          .addGap(5, 5, 5)
          .addComponent(jSeparator1, javax.swing.GroupLayout.PREFERRED_SIZE, 10, javax.swing.GroupLayout.PREFERRED_SIZE)
          .addGap(3, 3, 3)
          .addComponent(jLabel2)
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
          .addComponent(roomPasswordEntry, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
          .addComponent(roomPasswordSubmit)
          .addContainerGap())
      );

      pmSendButton.setLabel("Send");
      pmSendButton.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
          pmSendButtonActionPerformed(evt);
        }
      });

      pmLabel.setText("You are sending a private message to: ");

      javax.swing.GroupLayout privateMsgPanelLayout = new javax.swing.GroupLayout(privateMsgPanel);
      privateMsgPanel.setLayout(privateMsgPanelLayout);
      privateMsgPanelLayout.setHorizontalGroup(
        privateMsgPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(privateMsgPanelLayout.createSequentialGroup()
          .addContainerGap()
          .addGroup(privateMsgPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(privateMsgPanelLayout.createSequentialGroup()
              .addComponent(pmField, javax.swing.GroupLayout.PREFERRED_SIZE, 188, javax.swing.GroupLayout.PREFERRED_SIZE)
              .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
              .addComponent(pmSendButton, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addComponent(pmLabel))
          .addContainerGap(22, Short.MAX_VALUE))
      );
      privateMsgPanelLayout.setVerticalGroup(
        privateMsgPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(privateMsgPanelLayout.createSequentialGroup()
          .addGap(8, 8, 8)
          .addComponent(pmLabel)
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
          .addGroup(privateMsgPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
            .addComponent(pmField)
            .addComponent(pmSendButton, javax.swing.GroupLayout.PREFERRED_SIZE, 52, javax.swing.GroupLayout.PREFERRED_SIZE))
          .addContainerGap())
      );

      javax.swing.GroupLayout frontpageLayout = new javax.swing.GroupLayout(frontpage);
      frontpage.setLayout(frontpageLayout);
      frontpageLayout.setHorizontalGroup(
        frontpageLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
    .addComponent(announcments, javax.swing.GroupLayout.DEFAULT_SIZE, 444, Short.MAX_VALUE)
);
frontpageLayout.setVerticalGroup(
    frontpageLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addComponent(announcments, javax.swing.GroupLayout.DEFAULT_SIZE, 305, Short.MAX_VALUE)
);

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

connectionsScrollPanel.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
connectionsScrollPanel.addContainerListener(new java.awt.event.ContainerAdapter() {
    public void componentAdded(java.awt.event.ContainerEvent evt) {
        connectionsScrollPanelComponentAdded(evt);
    }
});

javax.swing.GroupLayout connectionsPanelLayout = new javax.swing.GroupLayout(connectionsPanel);
connectionsPanel.setLayout(connectionsPanelLayout);
connectionsPanelLayout.setHorizontalGroup(
    connectionsPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGap(0, 610, Short.MAX_VALUE)
);
connectionsPanelLayout.setVerticalGroup(
    connectionsPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGap(0, 291, Short.MAX_VALUE)
);

connectionsScrollPanel.setViewportView(connectionsPanel);

textLog.setEditable(false);

sendMessageButton.setText("-->");
sendMessageButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        sendMessageButtonActionPerformed(evt);
    }
});
sendMessageButton.addKeyListener(new java.awt.event.KeyAdapter() {
    public void keyPressed(java.awt.event.KeyEvent evt) {
        sendMessageButtonKeyPressed(evt);
    }
});

welcomeLabel.setText("Welcome to Conclave");

refreshConnections.setText("Refresh Connections");
refreshConnections.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        refreshConnectionsActionPerformed(evt);
    }
});

disconnectButton.setText("Disconnect");
disconnectButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        disconnectButtonActionPerformed(evt);
```

```
    }
});

leaveRoomButton.setText("Leave Room");
leaveRoomButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        leaveRoomButtonActionPerformed(evt);
    }
});

textInputArea.setColumns(20);
textInputArea.setFont(new java.awt.Font("Monospaced", 0, 14)); // NOI18N
textInputArea.setRows(5);
jScrollPane3.setViewportView(textInputArea);

exportChatLogButton.setText("Export ChatLog");
exportChatLogButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        exportChatLogButtonActionPerformed(evt);
    }
});

roomFilterCheckbox.setSelected(true);
roomFilterCheckbox.setText("View Room Messages");
roomFilterCheckbox.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        roomFilterCheckboxActionPerformed(evt);
    }
});

systemFilterCheckbox.setSelected(true);
systemFilterCheckbox.setText("View System Messages");
systemFilterCheckbox.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        systemFilterCheckboxActionPerformed(evt);
    }
});

privateFilterCheckbox.setSelected(true);
privateFilterCheckbox.setText("View Private Messages");
privateFilterCheckbox.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        privateFilterCheckboxActionPerformed(evt);
    }
});

adminFilterCheckbox.setSelected(true);
adminFilterCheckbox.setText("View Admin Messages");
adminFilterCheckbox.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        adminFilterCheckboxActionPerformed(evt);
    }
});

filterLabel.setText("Filter Chatlog messages:");
```

```
    javax.swing.GroupLayout chatlogFilterPanelLayout = new javax.swing.GroupLayout(chatlogFilterPanel);
    chatlogFilterPanel.setLayout(chatlogFilterPanelLayout);
    chatlogFilterPanelLayout.setHorizontalGroup(
      chatlogFilterPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(chatlogFilterPanelLayout.createSequentialGroup()
        .addContainerGap()
        .addGroup(chatlogFilterPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addComponent(systemFilterCheckbox)
          .addComponent(privateFilterCheckbox)
          .addComponent(adminFilterCheckbox)
          .addComponent(roomFilterCheckbox)
          .addComponent(filterLabel))
        .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
    );
    chatlogFilterPanelLayout.setVerticalGroup(
      chatlogFilterPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, chatlogFilterPanelLayout.createSequentialGroup()
        .addContainerGap()
        .addComponent(filterLabel)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 12, Short.MAX_VALUE)
        .addComponent(roomFilterCheckbox)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(systemFilterCheckbox)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(privateFilterCheckbox)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(adminFilterCheckbox))
    );

    interactablePanel.setMaximumSize(new java.awt.Dimension(1000, 800));
    tabbedPanel.addTab("Main", interactablePanel);

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
      layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(layout.createSequentialGroup()
        .addContainerGap()
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addComponent(refreshConnections, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
          .addComponent(disconnectButton, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
          .addComponent(leaveRoomButton, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
          .addComponent(exportChatLogButton, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
          .addComponent(chatlogFilterPanel, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
          .addComponent(connectionsScrollPanel, javax.swing.GroupLayout.DEFAULT_SIZE, 234, Short.MAX_VALUE)
          .addGroup(layout.createSequentialGroup()
            .addComponent(welcomeLabel)
            .addGap(0, 0, Short.MAX_VALUE)))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
```

```
                .addComponent(jScrollPane3)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(sendMessageButton))
            .addComponent(textLog, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
                .addGap(0, 0, Short.MAX_VALUE)
                .addComponent(tabbedPanel, javax.swing.GroupLayout.PREFERRED_SIZE, 745, javax.swing.GroupLayout.PREFERRED_SIZE)))
        .addContainerGap())
    );
    layout.setVerticalGroup(
      layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(layout.createSequentialGroup()
        .addGap(6, 6, 6)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
          .addGroup(layout.createSequentialGroup()
            .addComponent(welcomeLabel)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(connectionsScrollPanel))
          .addComponent(tabbedPanel))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
          .addGroup(layout.createSequentialGroup()
            .addComponent(refreshConnections, javax.swing.GroupLayout.PREFERRED_SIZE, 23,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(disconnectButton)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(leaveRoomButton)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(exportChatLogButton)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
            .addComponent(chatlogFilterPanel, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
          .addGroup(layout.createSequentialGroup()
            .addComponent(textLog, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
            .addGap(12, 12, 12)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING, false)
              .addComponent(jScrollPane3, javax.swing.GroupLayout.PREFERRED_SIZE, 0, Short.MAX_VALUE)
              .addComponent(sendMessageButton, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 54, Short.MAX_VALUE))))
        .addGap(17, 17, 17))
    );

    pack();
  }// </editor-fold>

  private void disconnectButtonActionPerformed(java.awt.event.ActionEvent evt) {
    try {
      if (inRoom) {
        if (client.getRoomType() == 2 && broadcastPanel != null) {
          broadcastPanel.stop();
        }
        client.leaveRoom();
      }
```

```
            client.disconnect();
            inRoom = false;
        } catch (RemoteException e) {
            e.printStackTrace();
        }
        System.exit(-1);
    }

    private void leaveRoomButtonActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            if (inRoom) {
                if (client.getRoomType() == 2 && broadcastPanel != null) {
                    broadcastPanel.stop();
                }
                client.leaveRoom();
                inRoom = false;
                buildFrontpage();
            } else {
                client.updateChatLog(new Message("System", client.getUsername(), "You are not in a room", 2));
            }
        } catch (RemoteException e) {

        }
    }

    private void exportChatLogButtonActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            String exportFileName = "ChatLog_" + client.getUsername() + lastExport++ + ".txt";
            FileWriter fileWriter = new FileWriter(exportFileName);
            BufferedWriter br = new BufferedWriter(fileWriter);
            ArrayList<Message> lineWriter = new ArrayList(client.getChatlogUpdates(0));
            String line = "";
            for (Message message : lineWriter) {
                line = message.msgDisplay();
                br.write(line);
                br.newLine();
            }
            br.close();
            client.updateChatLog(new Message("System", client.getUsername(), "You have exported a chatlog, saved as: " + exportFileName, 2));
        } catch (RemoteException e) {
            e.printStackTrace();
        } catch (IOException ex) {
            Logger.getLogger(SwingGUI.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    private void sendMessageButtonActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        try {
            if (inRoom) {
                client.postMessage(textInputArea.getText());
                textInputArea.setText("");
            }
        } catch (RemoteException e) {
            e.printStackTrace();
        }
```

```
}

private void refreshConnectionsActionPerformed(java.awt.event.ActionEvent evt) {
   try {
      refreshConnections();
   } catch (RemoteException e) {
      e.printStackTrace();
   }
}

private void connectionsScrollPanelComponentAdded(java.awt.event.ContainerEvent evt) {
   // TODO add your handling code here:
}

private void roomPasswordSubmitActionPerformed(java.awt.event.ActionEvent evt) {
   // TODO add your handling code here:
}

private void roomPasswordEntryActionPerformed(java.awt.event.ActionEvent evt) {
   // TODO add your handling code here:
}

private void pmSendButtonActionPerformed(java.awt.event.ActionEvent evt) {
   // TODO add your handling code here:
}

private void roomFilterCheckboxActionPerformed(java.awt.event.ActionEvent evt) {
   if (roomFilterCheckbox.isSelected()) {
      chatlogViewCategories.add("Room");
   } else {
      chatlogViewCategories.remove("Room");
   }
   resetChatlogView();
}

private void systemFilterCheckboxActionPerformed(java.awt.event.ActionEvent evt) {
   if (systemFilterCheckbox.isSelected()) {
      chatlogViewCategories.add("System");
   } else {
      chatlogViewCategories.remove("System");
   }
   resetChatlogView();
}

private void privateFilterCheckboxActionPerformed(java.awt.event.ActionEvent evt) {
   if (privateFilterCheckbox.isSelected()) {
      chatlogViewCategories.add("Private");
   } else {
      chatlogViewCategories.remove("Private");
   }
   resetChatlogView();
}

private void adminFilterCheckboxActionPerformed(java.awt.event.ActionEvent evt) {
   if (adminFilterCheckbox.isSelected()) {
      chatlogViewCategories.add("Admin");
```

```
    } else {
        chatlogViewCategories.remove("Admin");
    }
    resetChatlogView();
}

private void sendMessageButtonKeyPressed(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
}

private void resetChatlogView() {
    textLog.setText("");
    lastMessageLine = 0;
}

private void buildFrontpage() {
    interactablePanel.removeAll();
    interactablePanel.add(frontpage);
    updateFrontpage();
}

private void updateFrontpage() {
    try {
        announcments.removeAll();
        for (Announcement update : client.getFrontpage()) {
            String line = update.getName() + ": " + update.getText();
            announcments.add(line);
        }
    } catch (RemoteException ex) {
        Logger.getLogger(SwingGUI.class.getName()).log(Level.SEVERE, null, ex);
    }
}

private void interactConnection(java.awt.event.ActionEvent evt, String entryName) {
    try {
        if (!inRoom) {
            if (client.hasPassword(entryName)) {
                final JFrame newFrame = new JFrame();
                final String roomPasswordName = entryName;
                //newFrame.setPreferredSize(new Dimension(300, 400));
                //passwordFormPanel.setPreferredSize(new Dimension(300, 400));
                roomPasswordSubmit.addActionListener(new java.awt.event.ActionListener() {
                    @Override
                    public void actionPerformed(java.awt.event.ActionEvent evt) {
                        validatePassword(roomPasswordName, roomPasswordEntry.getText());
                        roomPasswordEntry.setText("");
                        newFrame.setVisible(false);
                    }
                });
                newFrame.add(passwordFormPanel);
                newFrame.pack();
                newFrame.setVisible(true);
            } else {
                joinRoom(entryName);
            }
        } else if (inRoom) {
```

```
            final JFrame newFrame = new JFrame();
            final String recipietUsername = entryName;
            final String senderUsername = client.getUsername();
            //newFrame.setPreferredSize(new Dimension(300, 400));
            //passwordFormPanel.setPreferredSize(new Dimension(300, 400));
            pmSendButton.addActionListener(new java.awt.event.ActionListener() {
               @Override
               public void actionPerformed(java.awt.event.ActionEvent evt) {
                  try {
                     client.sendPrivateMessage(pmField.getText(), recipietUsername);
                     pmField.setText("");
                     newFrame.setVisible(false);
                  } catch (RemoteException e) {
                     Message newMessage = new Message("System", senderUsername, "That user could not be reached", 2);
                     try {
                        client.updateChatLog(newMessage);
                     } catch (RemoteException ex) {
                        Logger.getLogger(SwingGUI.class.getName()).log(Level.SEVERE, null, ex);
                     }
                  }
               }
            });
            newFrame.add(privateMsgPanel);
            newFrame.pack();
            newFrame.setVisible(true);
         }
      } catch (RemoteException e) {
         e.printStackTrace();
      }
   }

   private void validatePassword(String entryName, String passwordEntry) {
      try {
         String msg = "Error";
         boolean validated = client.joinRoom(entryName, passwordEntry);
         if (validated) {
            msg = "Correct Password";
            interactablePanel.removeAll();
            inRoom = true;
            if (client.getRoomType() == 2) {
               buildBroadcastPanel();
            }
         } else {
            msg = "Incorrect password";
         }
         client.updateChatLog(new Message("System", client.getUsername(), msg, 2));
      } catch (RemoteException e) {

      }
   }

   private void joinRoom(String entryName) {
      try {
         boolean joined = client.joinRoom(entryName);
         if (joined) {
            interactablePanel.removeAll();
```

```
            inRoom = true;
            if (client.getRoomType() == 2) {
               buildBroadcastPanel();
            }
         } else {
            client.updateChatLog(new Message("System", client.getUsername(), "You cannot connect to that room", 2));
         }
      } catch (RemoteException e) {

      }
   }

   private void setConnectionsArea(ConnectionsLog connections) {
      connectionsPanel.removeAll();
      connectionsPanel.setLayout(new BoxLayout(connectionsPanel, BoxLayout.PAGE_AXIS));
      connectionsPanel.setBorder(BorderFactory.createEmptyBorder(0, 10, 10, 10));
      connectionsPanel.add(Box.createHorizontalGlue());
      connectionsPanel.add(Box.createRigidArea(new Dimension(10, 0)));
      int i = 0;
      for (ConnectionEntry entry : connections.getAllConnections()) {
         final String entryName = entry.getName();
         String label = i + ":" + entryName;
         JLabel connectionDesc = new JLabel(entry.getDesc());
         JButton newInteractionButton = new JButton(label);
         newInteractionButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
               interactConnection(evt, entryName);
            }
         });
         connectionsPanel.add(newInteractionButton);
         connectionsPanel.add(connectionDesc);
         connectionsPanel.add(new JSeparator(SwingConstants.HORIZONTAL));
         i++;
      }
      pack();
   }

   private void startUpdates() {
      Thread updates;
      updates = new Thread(new Runnable() {
         @Override
         public void run() {
            try {
               while (client.isConnected()) {
                  inRoom = client.inRoom();
                  if (client.hasConnectionsUpdated()) {
                     refreshConnections();
                  }
                  if (client.hasFrontpageUpdated()) {
                     updateFrontpage();
                  }
                  int lstMsgClient = client.getLastMessageLine();
                  if (lstMsgClient > lastMessageLine) {
                     LinkedList<Message> newMessages = client.getChatlogUpdates(lastMessageLine);
                     for (Message newMessage : newMessages) {
                        if (newMessage != null) {
```

```
                    System.out.println(newMessage.msgDisplay());
                    updateChatlog(newMessage);
                  }
                }
                lastMessageLine = lstMsgClient;
              }
              Thread.sleep(200);
            }
            updateChatlog(new Message("System", client.getUsername(), "You have been disconnected, Conclave will exit soon.", 2));
            if (inRoom) {
              if (client.getRoomType() == 2 && broadcastPanel != null) {
                broadcastPanel.stop();
              }
              client.leaveRoom();
            }
            inRoom = false;
            Thread.sleep(3000);
            System.exit(-1);
          } catch (RemoteException | InterruptedException e) {
          }
        }
      });
      updates.start();
    }

    private void updateChatlog(Message msg) {
      if (msg != null && chatlogViewCategories.contains(msg.getType())) {
        textLog.append(msg.msgDisplay());
        textLog.append("\n");
        pack();
      }
    }
    /**
     * @param args the command line arguments
     */

    // Variables declaration - do not modify
    private javax.swing.JCheckBox adminFilterCheckbox;
    private java.awt.List announcments;
    private javax.swing.JPanel chatlogFilterPanel;
    private javax.swing.JPanel connectionsPanel;
    private javax.swing.JScrollPane connectionsScrollPanel;
    private javax.swing.JButton disconnectButton;
    private javax.swing.JButton exportChatLogButton;
    private javax.swing.JLabel filterLabel;
    private javax.swing.ButtonGroup filtersMsg;
    private javax.swing.JPanel frontpage;
    private javax.swing.JPanel interactablePanel;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JMenu jMenu1;
    private javax.swing.JMenu jMenu2;
    private javax.swing.JMenuBar jMenuBar1;
    private javax.swing.JMenuItem jMenuItem1;
    private javax.swing.JScrollPane jScrollPane1;
```

```
    private javax.swing.JScrollPane jScrollPane3;
    private javax.swing.JSeparator jSeparator1;
    private javax.swing.JButton leaveRoomButton;
    private javax.swing.JPanel passwordFormPanel;
    private javax.swing.JTextField pmField;
    private javax.swing.JLabel pmLabel;
    private java.awt.Button pmSendButton;
    private javax.swing.JCheckBox privateFilterCheckbox;
    private javax.swing.JPanel privateMsgPanel;
    private javax.swing.JButton refreshConnections;
    private javax.swing.JCheckBox roomFilterCheckbox;
    private javax.swing.JTextField roomPasswordEntry;
    private javax.swing.JButton roomPasswordSubmit;
    private javax.swing.JButton sendMessageButton;
    private javax.swing.JCheckBox systemFilterCheckbox;
    private javax.swing.JTabbedPane tabbedPanel;
    private javax.swing.JTextArea textInputArea;
    private java.awt.TextArea textLog;
    private javax.swing.JLabel welcomeLabel;
    // End of variables declaration

    private void initilizeAdminTab() {
        AdminPanel adminPanel = new AdminPanel(client);
        tabbedPanel.add("Admin", adminPanel);
    }

    private void buildBroadcastPanel() {
        broadcastPanel = new BroadcastPanel(client);
        interactablePanel.removeAll();
        interactablePanel.add(broadcastPanel);
        pack();
    }

    private void refreshConnections() throws RemoteException {
        ConnectionsLog connectionsLog = client.viewAllConnections();
        setConnectionsArea(connectionsLog);
    }
}
```

DirectChannelBufferFactory class:

```
package conclaveclient.Conference;
//Taken from SARXOS webcam example
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import org.jboss.netty.buffer.ChannelBuffer;
import org.jboss.netty.buffer.ChannelBufferFactory;
import org.jboss.netty.buffer.ChannelBuffers;


public class DirectChannelBufferFactory implements ChannelBufferFactory{

        @Override
        public ChannelBuffer getBuffer(int capacity) {
                return ChannelBuffers.directBuffer(capacity);
        }

        @Override
```

```
	public ChannelBuffer getBuffer(ByteOrder endianness, int capacity) {
		return ChannelBuffers.directBuffer(endianness, capacity);
	}

	@Override
	public ChannelBuffer getBuffer(byte[] array, int offset, int length) {
		ChannelBuffer channelBuffer = ChannelBuffers.directBuffer(length);
		channelBuffer.writeBytes(array, offset, length);
		return channelBuffer;
	}

	@Override
	public ChannelBuffer getBuffer(ByteOrder endianness, byte[] array,
				int offset, int length) {
		ChannelBuffer channelBuffer = ChannelBuffers.directBuffer(endianness,length);
		channelBuffer.writeBytes(array, offset, length);
		return channelBuffer;
	}

	@Override
	public ChannelBuffer getBuffer(ByteBuffer nioBuffer) {
		int size = nioBuffer.capacity();
		ChannelBuffer channelBuffer = ChannelBuffers.directBuffer(nioBuffer.order(), size);
		channelBuffer.writeBytes(nioBuffer);
		return channelBuffer;
	}

	@Override
	public ByteOrder getDefaultOrder() {
		return ByteOrder.BIG_ENDIAN;
	}

}
```

ListeningClient class:

```
package conclaveclient.Conference;
//Taken from SARXOS webcam example
import conclaveclient.Conference.display.VideoPanel;
import conclaveclient.Conference.interfaces.StreamFrameListener;

import java.awt.Dimension;
import java.awt.image.BufferedImage;
import java.net.InetSocketAddress;

public class ListeningClient {

	private static VideoPanel displayWindow;
    private static StreamClientAgent clientAgent = null;

	public static void run(VideoPanel windowFrame, InetSocketAddress addr, Dimension d) {
		//setup the videoWindow
        displayWindow = windowFrame;
		//setup the connection
		//logger.info("setup dimension :{}",dimension);
		clientAgent = new StreamClientAgent(new StreamFrameListenerIMPL(),d);
		clientAgent.connect(addr);
	}
```

```
    public static void close()
    {
      if (clientAgent!=null)
      {
      clientAgent.stop();
      clientAgent = null;
      }
    }

    public static boolean isActive()
    {
      if (clientAgent != null)
      {
        return true;
      }
      return false;
    }

          protected static class StreamFrameListenerIMPL implements StreamFrameListener{
                  private volatile long count = 0;
                  @Override
                  public void onFrameReceived(BufferedImage image) {
                        //logger.info("frame received :{}",count++);
                        displayWindow.updateImage(image);
                  }

          }
}
```

SteramClientAgent class:

```
package conclaveclient.Conference;
//Taken from SARXOS webcam example
import conclaveclient.Conference.interfaces.StreamClientAgentInterface;
import conclaveclient.Conference.interfaces.StreamClientListener;
import conclaveclient.Conference.interfaces.StreamFrameListener;
import java.awt.Dimension;
import java.net.SocketAddress;
import java.util.concurrent.Executors;
import org.jboss.netty.bootstrap.ClientBootstrap;
import org.jboss.netty.channel.Channel;
import org.jboss.netty.channel.socket.nio.NioClientSocketChannelFactory;
public class StreamClientAgent implements StreamClientAgentInterface {

  //protected final static Logger logger = LoggerFactory.getLogger(StreamClientAgent.class);
  protected final ClientBootstrap clientBootstrap;
  protected final StreamClientListener streamClientListener;
  protected final StreamFrameListener streamFrameListener;
  protected final Dimension dimension;
  protected Channel clientChannel;

  public StreamClientAgent(StreamFrameListener streamFrameListener,
      Dimension dimension) {
    super();
    this.dimension = dimension;
    this.clientBootstrap = new ClientBootstrap();
    this.clientBootstrap.setFactory(new NioClientSocketChannelFactory(
```

```
            Executors.newCachedThreadPool(),
            Executors.newCachedThreadPool()));
        this.streamFrameListener = streamFrameListener;
        this.streamClientListener = new StreamClientListenerIMPL();
        this.clientBootstrap.setPipelineFactory(
            new StreamClientChannelPipelineFactory(
                streamClientListener,
                streamFrameListener,
                dimension));
    }

    @Override
    public void connect(SocketAddress streamServerAddress) {
        //logger.info("going to connect to stream server :{}", streamServerAddress);
        clientBootstrap.connect(streamServerAddress);
    }

    @Override
    public void stop() {
        clientChannel.close();
        clientBootstrap.releaseExternalResources();
    }

    protected class StreamClientListenerIMPL implements StreamClientListener {

        @Override
        public void onConnected(Channel channel) {
            //    logger.info("stream connected to server at :{}",channel);
            clientChannel = channel;
        }

        @Override
        public void onDisconnected(Channel channel) {
            //    logger.info("stream disconnected to server at :{}",channel);
        }

        @Override
        public void onException(Channel channel, Throwable t) {
            t.printStackTrace();
            //    logger.debug("exception at :{},exception :{}",channel,t);
        }

    }

}
```

StreamClientChannelPipelineFactory class:

```
package conclaveclient.Conference;
//Taken from SARXOS webcam example
import conclaveclient.Conference.interfaces.StreamClientListener;
import conclaveclient.Conference.interfaces.StreamFrameListener;
import conclaveclient.Handlers.utlity.H264StreamDecoder;
import conclaveclient.Handlers.StreamClientHandler;
import java.awt.Dimension;
import org.jboss.netty.channel.ChannelPipeline;
import org.jboss.netty.channel.ChannelPipelineFactory;
```

```
import org.jboss.netty.channel.Channels;
import org.jboss.netty.handler.codec.frame.LengthFieldBasedFrameDecoder;

public class StreamClientChannelPipelineFactory implements ChannelPipelineFactory {

   protected final StreamClientListener streamClientListener;
   protected final StreamFrameListener streamFrameListener;
   protected final Dimension dimension;

   public StreamClientChannelPipelineFactory(
         StreamClientListener streamClientListener,
         StreamFrameListener streamFrameListener, Dimension dimension) {
      super();
      this.streamClientListener = streamClientListener;
      this.streamFrameListener = streamFrameListener;
      this.dimension = dimension;
   }

   @Override
   public ChannelPipeline getPipeline() throws Exception {
      ChannelPipeline pipeline = Channels.pipeline();
      //Add the client handler, which will update the "VideoPanel" on the GUI.
      pipeline.addLast("stream client handler", new StreamClientHandler(streamClientListener));
      //Add a frame decoder, which will decode the frame recieved from a channel buffer.
      pipeline.addLast("frame decoder", new LengthFieldBasedFrameDecoder(Integer.MAX_VALUE, 0, 4, 0, 4));
      //Add a stream handler, which will decodes the stream from a h264 to a buffered image
      pipeline.addLast("stream handler", new H264StreamDecoder(streamFrameListener, dimension, false, false));
      return pipeline;
   }
}
```

StreamSeverAgent:

```
package conclaveclient.Conference;
//Taken from SARXOS webcam example
import com.github.sarxos.webcam.Webcam;
import conclaveclient.Conference.interfaces.StreamServerAgentInterface;
import conclaveclient.Conference.interfaces.StreamServerListener;
import conclaveclient.Handlers.utlity.H264StreamEncoder;
import java.awt.Dimension;
import java.awt.image.BufferedImage;
import java.net.SocketAddress;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.ScheduledThreadPoolExecutor;
import java.util.concurrent.TimeUnit;
import org.jboss.netty.bootstrap.ServerBootstrap;
import org.jboss.netty.channel.Channel;
import org.jboss.netty.channel.group.ChannelGroup;
import org.jboss.netty.channel.group.DefaultChannelGroup;
import org.jboss.netty.channel.socket.nio.NioServerSocketChannelFactory;

public class StreamServerAgent implements StreamServerAgentInterface {
   //protected final static Logger logger = LoggerFactory.getLogger(StreamServer.class);
   protected final Webcam webcam;
   protected final Dimension dimension;
```

```
protected final ChannelGroup channelGroup = new DefaultChannelGroup();
protected final ServerBootstrap serverBootstrap;
protected final H264StreamEncoder h264StreamEncoder;
protected volatile boolean isStreaming;
protected ScheduledExecutorService timeWorker;
protected ExecutorService encodeWorker;
protected int FPS = 25;
protected ScheduledFuture<?> imageGrabTaskFuture;

public StreamServerAgent(Webcam webcam, Dimension dimension) {
   super();
   this.webcam = webcam;
   this.dimension = dimension;
   this.serverBootstrap = new ServerBootstrap();
   this.serverBootstrap.setFactory(new NioServerSocketChannelFactory(
         Executors.newCachedThreadPool(),
         Executors.newCachedThreadPool()));
   this.serverBootstrap.setPipelineFactory(new StreamServerChannelPipelineFactory(
         new StreamServerListenerIMPL(),
         dimension));
   this.timeWorker = new ScheduledThreadPoolExecutor(1);
   this.encodeWorker = Executors.newSingleThreadExecutor();
   this.h264StreamEncoder = new H264StreamEncoder(dimension, false);
}

public int getFPS() {
   return FPS;
}

public void setFPS(int fPS) {
   FPS = fPS;
}

@Override
public void start(SocketAddress streamAddress) {
   //logger.info("Server started :{}",streamAddress);
   Channel channel = serverBootstrap.bind(streamAddress);
   channelGroup.add(channel);
}

@Override
public void stop() {
   //logger.info("server is stopping");
   channelGroup.close();
   timeWorker.shutdown();
   encodeWorker.shutdown();
   serverBootstrap.releaseExternalResources();
}

private class StreamServerListenerIMPL implements StreamServerListener {

   @Override
   public void onClientConnectedIn(Channel channel) {
      //This will initilize the stream, when the first client connects.
      //The first client is always the streamer though.
      channelGroup.add(channel);
```

```
    System.out.println("Client has connected to stream: " + channel.toString());
    if (!isStreaming) {
      //Create a new task that will get BufferedImages from webcam.
      Runnable imageGrabTask = new ImageGrabTask();
      //Creates a new task that will fill the buffer frame by frame.
      ScheduledFuture<?> imageGrabFuture
          = timeWorker.scheduleWithFixedDelay(imageGrabTask,
              0,
              1000 / FPS,
              TimeUnit.MILLISECONDS);
      //Set the tast future to the actual worker field
      imageGrabTaskFuture = imageGrabFuture;
      isStreaming = true;
    }
    //logger.info("current connected clients :{}",channelGroup.size());
  }

  @Override
  public void onClientDisconnected(Channel channel) {
    channelGroup.remove(channel);
    int size = channelGroup.size();
    //logger.info("current connected clients :{}",size);
    if (size == 1) {
      imageGrabTaskFuture.cancel(false);
      webcam.close();
      isStreaming = false;
    }
  }

  @Override
  public void onExcaption(Channel channel, Throwable t) {
    t.printStackTrace();
    channelGroup.remove(channel);
    channel.close();
    int size = channelGroup.size();
    //logger.info("current connected clients :{}",size);
    if (size == 1) {
      //cancel the task
      imageGrabTaskFuture.cancel(false);
      webcam.close();
      isStreaming = false;
    }
  }

  protected volatile long frameCount = 0;

  private class ImageGrabTask implements Runnable {

    @Override
    public void run() {
      //logger.info("image grabed ,count :{}",frameCount++);
      BufferedImage bufferedImage = webcam.getImage();
      //If adding a H264 encoder to the pipeline, uncomment next code.
      //channelGroup.write(bufferedImage);
      encodeWorker.execute(new EncodeTask(bufferedImage));
    }
```

```
    }

    private class EncodeTask implements Runnable {

      private final BufferedImage image;

      public EncodeTask(BufferedImage image) {
        super();
        this.image = image;
      }

      @Override
      public void run() {
        try {
          Object msg = h264StreamEncoder.encode(image);
          if (msg != null) {
            channelGroup.write(msg);
          }
        } catch (Exception e) {
          e.printStackTrace();
        }
      }

    }

  }

}
```

StreamServerChannelPipelineFactory class:

```
package conclaveclient.Conference;
import conclaveclient.Conference.interfaces.StreamServerListener;
import conclaveclient.Handlers.StreamServerHandler;
import java.awt.Dimension;
import org.jboss.netty.channel.ChannelPipeline;
import org.jboss.netty.channel.ChannelPipelineFactory;
import org.jboss.netty.channel.Channels;
import org.jboss.netty.handler.codec.frame.LengthFieldPrepender;
//Taken from SARXOS webcam example
public class StreamServerChannelPipelineFactory implements ChannelPipelineFactory{
        protected final StreamServerListener streamServerListener;
        protected final Dimension dimension;

        public StreamServerChannelPipelineFactory(
                        StreamServerListener streamServerListener, Dimension dimension) {
                super();
                this.streamServerListener = streamServerListener;
                this.dimension = dimension;
        }

    @Override
        public ChannelPipeline getPipeline() throws Exception {
                ChannelPipeline pipeline = Channels.pipeline();

                //comment the netty's frame encoder ,if want to use the build in h264 encoder
                pipeline.addLast("frame encoder", new LengthFieldPrepender(4));
```

```
                pipeline.addLast("stream server handler", new StreamServerHandler(streamServerListener));
        //Uncomment if using local h264 encoder in pipeline.
                //pipeline.addLast("stream h264 encoder", new H264StreamEncoder(dimension,false));
                return pipeline;
    }

}
```

StreamingServer class:

```
package conclaveclient.Conference;

import com.github.sarxos.webcam.Webcam;
import com.github.sarxos.webcam.WebcamResolution;
import java.awt.Dimension;
import java.net.InetAddress;
import java.net.InetSocketAddress;
import java.net.UnknownHostException;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.lang.Exception;

/**
 *
 * @author BradleyW
 */
public class StreamingServer {

  private Webcam webcam = null;
  private Dimension dimension = null;
  StreamServerAgent serverAgent = null;
  InetSocketAddress socketAddress;

  public StreamingServer() throws Exception{
    try {
      socketAddress = new InetSocketAddress(InetAddress.getLocalHost(), 20000);
    } catch (UnknownHostException ex) {
      Logger.getLogger(StreamingServer.class.getName()).log(Level.SEVERE, null, ex);
    }
    if (Webcam.getWebcams()!=null)
    {
      webcam = Webcam.getDefault();
      dimension = WebcamResolution.VGA.getSize();
      webcam.setViewSize(dimension);
      Webcam.setAutoOpenMode(true);
    } else {
      throw new Exception();
    }
  }

  public void streamWebcam() {
    serverAgent = new StreamServerAgent(webcam, WebcamResolution.VGA.getSize());
    serverAgent.start(socketAddress);
  }

  public boolean webcamIsActive() {
    boolean active = false;
    if (webcam != null) {
```

```
            active = webcam.isOpen();
        }
        return active;
    }

    public String getName() {
        return webcam.getName();
    }

    public Dimension getDimension() {
        return dimension;
    }

    public InetSocketAddress getSocketIp() {
        return socketAddress;
    }

    public void stopStreaming()
    {
        if (serverAgent!=null)
        {
            serverAgent.stop();
            webcam.close();
        }
    }

}
```

BroadcastPanel class:

```
package conclaveclient.Conference.display;

import com.github.sarxos.webcam.Webcam;
import model.Message;
import conclaveclient.Conference.ListeningClient;
import conclaveclient.Conference.StreamingServer;
import conclaveinterfaces.IUserInterface;
import java.awt.Dimension;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.net.InetSocketAddress;
import java.rmi.RemoteException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.imageio.ImageIO;

/**
 *
 * @author BradleyW
 */
public class BroadcastPanel extends javax.swing.JPanel {

    /**
     * Creates new form BroadcastPanel
     */
    public static int lastPhotoID;
```

```
public BroadcastPanel(IUserInterface ui) {
    initComponents();
    videoContainer.setPreferredSize(new Dimension(600, 400)); //This is to keep the container from collapsing.
    streamerName.setText("To stream, hit the stream button in the toolbar.");
    client = ui;
    streaming = false;
    active = true;
    lastPhotoID = this.hashCode();
    openPanel();
    subscribeBroadcasterUpdates();
}

public void openPanel() {
    active = true;
    videoPanel.setVisible(true);
}

public void emptyPanel() {
    active = false;
    videoPanel.setVisible(false);
}

private IUserInterface client;
private boolean active;
private StreamingServer streamingServer;
private boolean streaming;

/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    videoContainer = new javax.swing.JPanel();
    videoPanel = new conclaveclient.Conference.display.VideoPanel();
    buttonsPanel = new javax.swing.JPanel();
    buttonA = new javax.swing.JButton();
    buttonC = new javax.swing.JButton();
    buttonB = new javax.swing.JButton();
    buttonD = new javax.swing.JButton();
    streamerName = new javax.swing.JLabel();

    videoContainer.setBorder(javax.swing.BorderFactory.createEtchedBorder());

    javax.swing.GroupLayout videoPanelLayout = new javax.swing.GroupLayout(videoPanel);
    videoPanel.setLayout(videoPanelLayout);
    videoPanelLayout.setHorizontalGroup(
        videoPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGap(0, 675, Short.MAX_VALUE)
    );
    videoPanelLayout.setVerticalGroup(
        videoPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGap(0, 414, Short.MAX_VALUE)
```

```
);

javax.swing.GroupLayout videoContainerLayout = new javax.swing.GroupLayout(videoContainer);
videoContainer.setLayout(videoContainerLayout);
videoContainerLayout.setHorizontalGroup(
  videoContainerLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
  .addGroup(videoContainerLayout.createSequentialGroup()
    .addContainerGap()
    .addComponent(videoPanel, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
    .addContainerGap())
);
videoContainerLayout.setVerticalGroup(
  videoContainerLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
  .addGroup(videoContainerLayout.createSequentialGroup()
    .addContainerGap()
    .addComponent(videoPanel, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
    .addContainerGap())
);

buttonA.setText("Start Streaming");
buttonA.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    buttonAActionPerformed(evt);
  }
});

buttonC.setText("Hide/Show Streamer Panel");
buttonC.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    buttonCActionPerformed(evt);
  }
});

buttonB.setText("Stop Streaming");
buttonB.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    buttonBActionPerformed(evt);
  }
});

buttonD.setText("Screenshot");
buttonD.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    buttonDActionPerformed(evt);
  }
});

javax.swing.GroupLayout buttonsPanelLayout = new javax.swing.GroupLayout(buttonsPanel);
buttonsPanel.setLayout(buttonsPanelLayout);
buttonsPanelLayout.setHorizontalGroup(
  buttonsPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
  .addGroup(buttonsPanelLayout.createSequentialGroup()
    .addContainerGap()
    .addGroup(buttonsPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
      .addComponent(buttonA, javax.swing.GroupLayout.DEFAULT_SIZE, 230, Short.MAX_VALUE)
      .addComponent(buttonC, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
```

```
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 129, Short.MAX_VALUE)
            .addGroup(buttonsPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
              .addComponent(buttonB, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.PREFERRED_SIZE, 227,
javax.swing.GroupLayout.PREFERRED_SIZE)
              .addComponent(buttonD, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.PREFERRED_SIZE, 227,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addContainerGap())
      );
      buttonsPanelLayout.setVerticalGroup(
        buttonsPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(buttonsPanelLayout.createSequentialGroup()
          .addContainerGap()
          .addGroup(buttonsPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(buttonA)
            .addComponent(buttonB))
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
          .addGroup(buttonsPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(buttonC)
            .addComponent(buttonD))
          .addContainerGap(12, Short.MAX_VALUE))
      );

      streamerName.setText("jLabel1");

      javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);
      this.setLayout(layout);
      layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
          .addGap(42, 42, 42)
          .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(streamerName)
            .addComponent(videoContainer, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
          .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
          .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
          .addComponent(buttonsPanel, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
          .addGap(56, 56, 56))
      );
      layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
          .addContainerGap()
          .addComponent(streamerName)
          .addGap(7, 7, 7)
          .addComponent(videoContainer, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
          .addComponent(buttonsPanel, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
          .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
      );
    }// </editor-fold>
```

```
  private void buttonBActionPerformed(java.awt.event.ActionEvent evt) {
    try {
      if (client.isConferenceStreaming()) {
        if (streaming) {
          client.updateChatLog(new Message("System", client.getUsername(), "You have stopped streaming", 2));
          client.stopBroadcasting();
          stop();
          emptyPanel();
        } else {
          try {
            client.updateChatLog(new Message("System", client.getUsername(), "You are not streaming.", 2));
          } catch (RemoteException ex) {
            Logger.getLogger(BroadcastPanel.class.getName()).log(Level.SEVERE, null, ex);
          }
        }
      } else {
        try {
          client.updateChatLog(new Message("System", client.getUsername(), "There is no stream active..", 2));
        } catch (RemoteException ex) {
          Logger.getLogger(BroadcastPanel.class.getName()).log(Level.SEVERE, null, ex);
        }
      }
    } catch (RemoteException e) {

    }
  }

  private void buttonAActionPerformed(java.awt.event.ActionEvent evt) {
    try {
      if (Webcam.getWebcams() != null) {
        for (Webcam cam : Webcam.getWebcams()) {
          client.updateChatLog(new Message("System", client.getUsername(), "Found webcam: " + cam.getName() + ", trying to initiate
Conclave Conference..", 2));
        }
        if (!client.isConferenceStreaming()) {
          try {
            String camName = startStreaming();
            client.broadcastToConference(getIP(), getDimension());
            client.recievePrivateMessage(new Message("System", client.getUsername(), "Webcam found: " + camName, 2));
          } catch (Exception e)
          {
            client.recievePrivateMessage(new Message("System", client.getUsername(), "A suitable webcam was not found.", 2));
          }
        } else {
          client.recievePrivateMessage(new Message("System", client.getUsername(), "A user is already streaming.", 2));
        }
      } else {
        client.updateChatLog(new Message("System", client.getUsername(), "Cannot detect a webcam.", 2));
      }
    } catch (RemoteException ex) {
    }
  }

  private void buttonCActionPerformed(java.awt.event.ActionEvent evt) {
    if (active) {
      emptyPanel();
```

```
            try {
                client.updateChatLog(new Message("System", client.getUsername(), "Hiding panel..", 2));
            } catch (RemoteException ex) {
                Logger.getLogger(BroadcastPanel.class.getName()).log(Level.SEVERE, null, ex);
            }
        } else {
            openPanel();
            try {
                client.updateChatLog(new Message("System", client.getUsername(), "Showing panel..", 2));
            } catch (RemoteException ex) {
                Logger.getLogger(BroadcastPanel.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }

    private void buttonDActionPerformed(java.awt.event.ActionEvent evt) {
        BufferedImage bi = videoPanel.image;
        String fileName = "Conclave_Screenshot" + lastPhotoID++ + ".png";
        File outputfile = new File(fileName);
        try {
            ImageIO.write(bi, "png", outputfile);
            client.updateChatLog(new Message("System", client.getUsername(), "You have taken a screenshot! Saved as: " + fileName, 2));
        } catch (IOException ex) {
            Logger.getLogger(BroadcastPanel.class.getName()).log(Level.SEVERE, null, ex);
        }
    }


    // Variables declaration - do not modify
    private javax.swing.JButton buttonA;
    private javax.swing.JButton buttonB;
    private javax.swing.JButton buttonC;
    private javax.swing.JButton buttonD;
    private javax.swing.JPanel buttonsPanel;
    private javax.swing.JLabel streamerName;
    private javax.swing.JPanel videoContainer;
    private conclaveclient.Conference.display.VideoPanel videoPanel;
    // End of variables declaration

    public void subscribeBroadcasterUpdates() {
        Thread roomUpdates = new Thread(new Runnable() {
            public void run() {
                try {
                    while (client.inRoom()) {
                        if (client.hasStreamerUpdated()) {
                            if (client.isConferenceStreaming()) {
                                if (!streaming) {
                                    listenToStream(client.getStreamerLocation(), client.getConferenceDimension(), client.getStreamerName());
                                    openPanel();

                                    try {
                                        client.updateChatLog(new Message("System", client.getUsername(), "A stream has started.", 2));
                                    } catch (RemoteException ex) {
                                        Logger.getLogger(BroadcastPanel.class.getName()).log(Level.SEVERE, null, ex);
                                    }
                                }
```

```
                    } else {
                        emptyPanel();
                        try {
                            client.updateChatLog(new Message("System", client.getUsername(), "A stream has stopped.", 2));
                        } catch (RemoteException ex) {
                            Logger.getLogger(BroadcastPanel.class.getName()).log(Level.SEVERE, null, ex);
                        }
                    }
                }
                try {
                    Thread.sleep(500);
                } catch (InterruptedException ex) {
                    Logger.getLogger(BroadcastPanel.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
            stop();
        } catch (RemoteException ex) {
            Logger.getLogger(BroadcastPanel.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
  });
  roomUpdates.start();
}

public void listenToStream(InetSocketAddress loc, Dimension d, String streamerName) throws RemoteException {
  this.streamerName.setText(streamerName + " is currently streaming.");
  openPanel();
  Dimension containerDimension = new Dimension(d.width + 20, d.height + 20);
  videoContainer.setPreferredSize(containerDimension);
  videoContainer.setMinimumSize(containerDimension);
  ListeningClient.run(videoPanel, loc, d);
  subscribeBroadcasterUpdates();
}

public String startStreaming() throws Exception{
  streamingServer = new StreamingServer();
  streamingServer.streamWebcam();
  streaming = true;
  try {
    listenToStream(getIP(), getDimension(), "You");
  } catch (RemoteException ex) {
    Logger.getLogger(BroadcastPanel.class.getName()).log(Level.SEVERE, null, ex);
  }
  return streamingServer.getName();
}

public void stop() {
  emptyPanel();
  if (active) {
    ListeningClient.close();
  }
  if (streaming) {
    streamingServer.stopStreaming();
    streaming = false;
  }
  streamerName.setText("To stream, hit the stream button in the toolbar.");
```

```
  }

  public InetSocketAddress getIP() {
    return streamingServer.getSocketIp();
  }

  public Dimension getDimension() {
    return streamingServer.getDimension();
  }
}
```

VideoPanel class:

```
package conclaveclient.Conference.display;
//Taken from SARXOS webcam example
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.FontMetrics;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.image.BufferedImage;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

import javax.swing.JPanel;

public class VideoPanel extends JPanel {

  /**
   *
   */
  private static final long serialVersionUID = -7292145875292244144L;

  protected BufferedImage image;
  protected final ExecutorService worker = Executors.newSingleThreadExecutor();

  @Override
  protected void paintComponent(Graphics g) {
    Graphics2D g2 = (Graphics2D) g;
    if (image == null) {
      g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
      g2.setBackground(Color.BLACK);
      g2.fillRect(0, 0, getWidth(), getHeight());

      int cx = (getWidth() - 70) / 2;
      int cy = (getHeight() - 40) / 2;

      g2.setStroke(new BasicStroke(2));
      g2.setColor(Color.LIGHT_GRAY);
      g2.fillRoundRect(cx, cy, 70, 40, 10, 10);
      g2.setColor(Color.WHITE);
      g2.fillOval(cx + 5, cy + 5, 30, 30);
      g2.setColor(Color.LIGHT_GRAY);
      g2.fillOval(cx + 10, cy + 10, 20, 20);
      g2.setColor(Color.WHITE);
      g2.fillOval(cx + 12, cy + 12, 16, 16);
      g2.fillRoundRect(cx + 50, cy + 5, 15, 10, 5, 5);
```

```
        g2.fillRect(cx + 63, cy + 25, 7, 2);
        g2.fillRect(cx + 63, cy + 28, 7, 2);
        g2.fillRect(cx + 63, cy + 31, 7, 2);

        g2.setColor(Color.DARK_GRAY);
        g2.setStroke(new BasicStroke(3));
        g2.drawLine(0, 0, getWidth(), getHeight());
        g2.drawLine(0, getHeight(), getWidth(), 0);

        String str = image == null ? "Connecting To Server" : "No Image";
        FontMetrics metrics = g2.getFontMetrics(getFont());
        int w = metrics.stringWidth(str);
        int h = metrics.getHeight();

        g2.setColor(Color.WHITE);
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_OFF);
        g2.drawString(str, (getWidth() - w) / 2, cy - h);

      } else {
        int width = image.getWidth();
        int height = image.getHeight();
        g2.clearRect(0, 0, width, height);
        g2.drawImage(image, 0, 0, width, height, null);
        setSize(width, height);
      }
    }

    public void updateImage(final BufferedImage update) {
      worker.execute(new Runnable() {

        @Override
        public void run() {
          image = update;
          repaint();
          //revalidate();
        }
      });
    }

    public void close() {
      worker.shutdown();
    }


    public boolean isActive() {
      return !worker.isShutdown();
    }

}
```

StreamClientAgentInterface class:

```
package conclaveclient.Conference.interfaces;

import java.net.SocketAddress;


public interface StreamClientAgentInterface {
```

```
	public void connect(SocketAddress streamServerAddress); //Connects to the stream via a SocketAddress
	public void stop(); //Stop recieving.
}
```

StreamClientListener class:

```
package conclaveclient.Conference.interfaces;

import org.jboss.netty.channel.Channel;

public interface StreamClientListener {
	public void onConnected(Channel channel); //Called when connected.
	public void onDisconnected(Channel channel); //Called when disconnected.
	public void onException(Channel channel,Throwable t); //Called when exception is thrown.
}
```

StreamFrameListener class:

```
package conclaveclient.Conference.interfaces;

import java.awt.image.BufferedImage;

public interface StreamFrameListener {
	public void onFrameReceived(BufferedImage image); //Called when the client recieves a frame.
}
```

StreamServerAgentInterface class:

```
package conclaveclient.Conference.interfaces;

import java.net.SocketAddress;

public interface StreamServerAgentInterface {
	public void start(SocketAddress streamAddress); //Called when the streamer begins streaming
	public void stop(); //Stops the streamers stream and kicks all connections.
}
```

StreamServerListener class:

```
package conclaveclient.Conference.interfaces;

import org.jboss.netty.channel.Channel;

public interface StreamServerListener {
	public void onClientConnectedIn(Channel channel); //Called on streamer side when client connects
	public void onClientDisconnected(Channel channel); //called on streamer side when client disconnects
	public void onExcaption(Channel channel,Throwable t); //When an exception is called.
}
```

DirectChannelBufferFactory class:

```
package conclaveclient.Handlers;
//Taken from SARXOS webcam example
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import org.jboss.netty.buffer.ChannelBuffer;
import org.jboss.netty.buffer.ChannelBufferFactory;
import org.jboss.netty.buffer.ChannelBuffers;

public class DirectChannelBufferFactory implements ChannelBufferFactory{

	@Override
	public ChannelBuffer getBuffer(int capacity) {
		return ChannelBuffers.directBuffer(capacity);
```

```
	}

	@Override
	public ChannelBuffer getBuffer(ByteOrder endianness, int capacity) {
		return ChannelBuffers.directBuffer(endianness, capacity);
	}

	@Override
	public ChannelBuffer getBuffer(byte[] array, int offset, int length) {
		ChannelBuffer channelBuffer = ChannelBuffers.directBuffer(length);
		channelBuffer.writeBytes(array, offset, length);
		return channelBuffer;
	}

	@Override
	public ChannelBuffer getBuffer(ByteOrder endianness, byte[] array,
				int offset, int length) {
		ChannelBuffer channelBuffer = ChannelBuffers.directBuffer(endianness,length);
		channelBuffer.writeBytes(array, offset, length);
		return channelBuffer;
	}

	@Override
	public ChannelBuffer getBuffer(ByteBuffer nioBuffer) {
		int size = nioBuffer.capacity();
		ChannelBuffer channelBuffer = ChannelBuffers.directBuffer(nioBuffer.order(), size);
		channelBuffer.writeBytes(nioBuffer);
		return channelBuffer;
	}

	@Override
	public ByteOrder getDefaultOrder() {
		return ByteOrder.BIG_ENDIAN;
	}
}
```

StreamClientHandler class:

```
package conclaveclient.Handlers;
//Taken from SARXOS webcam example
import conclaveclient.Handlers.utlity.FrameDecoder;
import conclaveclient.Conference.interfaces.StreamClientListener;
import org.jboss.netty.buffer.ChannelBuffer;
import org.jboss.netty.channel.Channel;
import org.jboss.netty.channel.ChannelHandlerContext;
import org.jboss.netty.channel.ChannelStateEvent;
import org.jboss.netty.channel.ExceptionEvent;
import org.jboss.netty.channel.MessageEvent;
import org.jboss.netty.channel.SimpleChannelHandler;

public class StreamClientHandler extends SimpleChannelHandler{
	protected final StreamClientListener streamClientListener;
	//protected final static Logger logger = LoggerFactory.getLogger(StreamClientHandler.class);
	protected final FrameDecoder frameDecoder = new FrameDecoder(4);
	public StreamClientHandler(StreamClientListener streamClientListener) {
		super();
		this.streamClientListener = streamClientListener;
	}
```

```
        @Override
        public void exceptionCaught(ChannelHandlerContext ctx, ExceptionEvent e)
                        throws Exception {
                Channel channel = e.getChannel();
                Throwable t = e.getCause();
                //logger.debug("exception at :{}",channel);
                streamClientListener.onException(channel, t);
                //super.exceptionCaught(ctx, e);
        }

        @Override
        public void channelConnected(ChannelHandlerContext ctx, ChannelStateEvent e)
                        throws Exception {
                Channel channel = e.getChannel();
                //logger.info("channel connected at {}",channel);
                streamClientListener.onConnected(channel);
                super.channelConnected(ctx, e);
        }

        @Override
        public void channelDisconnected(ChannelHandlerContext ctx,
                        ChannelStateEvent e) throws Exception {
                Channel channel = e.getChannel();
                //logger.info("channel disconnected at :{}",channel);
                streamClientListener.onDisconnected(channel);
                super.channelDisconnected(ctx, e);
        }

        @Override
        public void messageReceived(ChannelHandlerContext ctx, MessageEvent e)
                        throws Exception {
                ChannelBuffer channelBuffer = (ChannelBuffer)e.getMessage();
                //logger.info("message received :{}",channelBuffer.readableBytes());
                super.messageReceived(ctx, e);
        }
}
```

StreamServerHandler class:

```
package conclaveclient.Handlers;
//Taken from SARXOS webcam example
import conclaveclient.Conference.interfaces.StreamServerListener;
import org.jboss.netty.channel.Channel;
import org.jboss.netty.channel.ChannelHandlerContext;
import org.jboss.netty.channel.ChannelStateEvent;
import org.jboss.netty.channel.ExceptionEvent;
import org.jboss.netty.channel.SimpleChannelHandler;
import org.jboss.netty.channel.WriteCompletionEvent;

public class StreamServerHandler extends SimpleChannelHandler {

  protected final StreamServerListener streamServerListener;
  //protected final static Logger logger = LoggerFactory.getLogger(StreamServerHandler.class);

  public StreamServerHandler(StreamServerListener streamServerListener) {
    super();
    this.streamServerListener = streamServerListener;
```

```
    }

    @Override
    public void exceptionCaught(ChannelHandlerContext ctx, ExceptionEvent e)
            throws Exception {
        Channel channel = e.getChannel();
        Throwable t = e.getCause();
        //logger.debug("exception caught at :{},exception :{}", channel, t);
        streamServerListener.onExcaption(channel, t);
        //super.exceptionCaught(ctx, e);
    }

    @Override
    public void channelConnected(ChannelHandlerContext ctx, ChannelStateEvent e)
            throws Exception {
        Channel channel = e.getChannel();
        //logger.info("channel connected :{}", channel);
        streamServerListener.onClientConnectedIn(channel);
        super.channelConnected(ctx, e);
    }

    @Override
    public void channelDisconnected(ChannelHandlerContext ctx,
            ChannelStateEvent e) throws Exception {
        Channel channel = e.getChannel();
        //logger.info("channel disconnected :{}", channel);
        streamServerListener.onClientDisconnected(channel);
        super.channelDisconnected(ctx, e);
    }

    @Override
    public void writeComplete(ChannelHandlerContext ctx, WriteCompletionEvent e)
            throws Exception {
        Channel channel = e.getChannel();
        long size = e.getWrittenAmount();
        //logger.info("frame send at :{} size :{}", channel, size);
        super.writeComplete(ctx, e);
    }

}
```

FrameDecoder class:

```
package conclaveclient.Handlers.utlity;
//Taken from SARXOS webcam example
import org.jboss.netty.buffer.ChannelBuffer;
import org.jboss.netty.buffer.ChannelBuffers;
import org.jboss.netty.buffer.HeapChannelBufferFactory;


public class FrameDecoder{
        //protected final ChannelBuffer dataSink = ChannelBuffers.dynamicBuffer(65536);
        protected ChannelBuffer dataSink = ChannelBuffers.dynamicBuffer(65536, new HeapChannelBufferFactory());
        protected final int headLength;

        public FrameDecoder(int headLength) {
                super();
                this.headLength = headLength;
```

```
        }

        public ChannelBuffer decode(ChannelBuffer channelBuffer) throws Exception {
                dataSink.writeBytes(channelBuffer);
    int actualLengthFieldOffset = dataSink.readerIndex();
    long frameLength;
    switch (headLength) {
            case 1:
              frameLength = dataSink.getUnsignedByte(actualLengthFieldOffset);
              break;
            case 2:
              frameLength = dataSink.getUnsignedShort(actualLengthFieldOffset);
              break;
            case 3:
              frameLength = dataSink.getUnsignedMedium(actualLengthFieldOffset);
              break;
            case 4:
              frameLength = dataSink.getUnsignedInt(actualLengthFieldOffset);
              break;
            case 8:
              frameLength = dataSink.getLong(actualLengthFieldOffset);

              break;
            default:
              throw new Error("Incorrect frame type when decoding.");
    }


    if (frameLength < 0) {
      dataSink.skipBytes(headLength);
    }
    int frameLengthInt = (int) frameLength;
    final ChannelBuffer frame;
    if (dataSink.readableBytes() >= frameLengthInt + headLength) {
        dataSink.skipBytes(headLength);
        frame = ChannelBuffers.buffer(frameLengthInt);
        dataSink.readBytes(frame);
        dataSink.discardReadBytes();
                }else{
                        return null;
                }
                return frame;
        }
}
```

FrameEncoder class:

```
//Taken from SARXOS webcam example
package conclaveclient.Handlers.utlity;

import org.jboss.netty.buffer.ChannelBuffer;
import org.jboss.netty.buffer.ChannelBuffers;

public class FrameEncoder{
        protected final int headLength;

        public FrameEncoder(int headLength) {
```

```
                super();
                this.headLength = headLength;
        }


        public ChannelBuffer encode(ChannelBuffer channelBuffer) throws Exception {
                int length = channelBuffer.readableBytes();
                //System.out.println("message length :"+length);
                ChannelBuffer header = ChannelBuffers.buffer(headLength);
                //System.out.println(channelBuffer.order());
                switch (headLength) {
            case 1:
              if (length >= 256) {
                throw new IllegalArgumentException(
                        "length does not fit into a byte: " + length);
              }
              header.writeByte((byte) length);
              break;
            case 2:
              if (length >= 65536) {
                throw new IllegalArgumentException(
                        "length does not fit into a short integer: " + length);
              }
              header.writeShort((short) length);
              break;
            case 3:
              if (length >= 16777216) {
                throw new IllegalArgumentException(
                        "length does not fit into a medium integer: " + length);
              }
              header.writeMedium(length);
              break;
            case 4:
              header.writeInt(length);
              break;
            case 8:
              header.writeLong(length);
              break;
            default:
              throw new Error("Incorrect frame length when encoding.");
            }
                return ChannelBuffers.wrappedBuffer(header,channelBuffer);
        }

}
```

H264StreamDecoder class:

```
package conclaveclient.Handlers.utlity;
import conclaveclient.Conference.interfaces.StreamFrameListener;

import java.awt.Dimension;
import java.awt.image.BufferedImage;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

import org.jboss.netty.buffer.ChannelBuffer;
import org.jboss.netty.channel.Channel;
```

```
import org.jboss.netty.channel.ChannelHandlerContext;
import org.jboss.netty.handler.codec.oneone.OneToOneDecoder;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import com.xuggle.ferry.IBuffer;
import com.xuggle.xuggler.ICodec;
import com.xuggle.xuggler.IPacket;
import com.xuggle.xuggler.IPixelFormat;
import com.xuggle.xuggler.IStreamCoder;
import com.xuggle.xuggler.IStreamCoder.Direction;
import com.xuggle.xuggler.IVideoPicture;
import com.xuggle.xuggler.video.ConverterFactory;
import com.xuggle.xuggler.video.ConverterFactory.Type;
import com.xuggle.xuggler.video.IConverter;


/**
 * This codec will encode the bufferedImage to h264 stream
**/
//Taken from SARXOS webcam example

public class H264StreamDecoder extends OneToOneDecoder {

	protected final static Logger logger = LoggerFactory.getLogger(H264StreamDecoder.class);
	protected final IStreamCoder iStreamCoder = IStreamCoder.make(Direction.DECODING, ICodec.ID.CODEC_ID_H264);
	protected final Type type = ConverterFactory.findRegisteredConverter(ConverterFactory.XUGGLER_BGR_24);
	protected final StreamFrameListener streamFrameListener;
	protected final Dimension dimension;

	protected final FrameDecoder frameDecoder;
	protected final ExecutorService decodeWorker;

	/**
	 *
	 * @param streamFrameListener
	 * @param dimension
	 * @param internalFrameDecoder
	 * @param decodeInOtherThread
	 */

	public H264StreamDecoder(StreamFrameListener streamFrameListener,
			Dimension dimension, boolean internalFrameDecoder,
			boolean decodeInOtherThread) {

			super();

			this.streamFrameListener = streamFrameListener;
			this.dimension = dimension;
			if (internalFrameDecoder) {
				frameDecoder = new FrameDecoder(4);
			} else {
				frameDecoder = null;
			}
			if (decodeInOtherThread) {
				decodeWorker = Executors.newSingleThreadExecutor();
			} else {
```

```
                decodeWorker = null;
            }

            initialize();
        }

        private void initialize() {
            iStreamCoder.open(null, null);
        }

        @Override
        protected Object decode(ChannelHandlerContext ctx, Channel channel, final Object msg) throws Exception {

            if (decodeWorker != null) {
                decodeWorker.execute(new decodeTask(msg));
                return null;
            } else {
                if (msg == null) {
                    throw new NullPointerException("you cannot pass into an null to the decode");
                }
                ChannelBuffer frameBuffer;
                if (frameDecoder != null) {
                    frameBuffer = frameDecoder.decode((ChannelBuffer) msg);
                    if (frameBuffer == null) {
                        return null;
                    }

                } else {
                    frameBuffer = (ChannelBuffer) msg;
                }

                int size = frameBuffer.readableBytes();
                logger.info("decode the frame size :{}", size);
                // start to decode
                IBuffer iBuffer = IBuffer.make(null, size);
                IPacket iPacket = IPacket.make(iBuffer);
                iPacket.getByteBuffer().put(frameBuffer.toByteBuffer());
                // decode the packet
                if (!iPacket.isComplete()) {
                    return null;
                }

                IVideoPicture picture = IVideoPicture.make(IPixelFormat.Type.YUV420P,
                    dimension.width, dimension.height);
                try {
                    // decode the packet into the video picture
                    int postion = 0;
                    int packageSize = iPacket.getSize();
                    while (postion < packageSize) {
                        postion += iStreamCoder.decodeVideo(picture, iPacket, postion);
                        if (postion < 0) {
                            throw new RuntimeException("error "
                                    + " decoding video");
                        }
                        // if this is a complete picture, dispatch the picture
                        if (picture.isComplete()) {
```

```
                                        IConverter converter = ConverterFactory.createConverter(type
                                                .getDescriptor(), picture);
                                        BufferedImage image = converter.toImage(picture);
                                        // here ,put out the image
                                        if (streamFrameListener != null) {
                                                streamFrameListener.onFrameReceived(image);
                                        }
                                        converter.delete();
                                } else {
                                        picture.delete();
                                        iPacket.delete();
                                        return null;
                                }
                                // clean the picture and reuse it
                                picture.getByteBuffer().clear();
                        }
                } finally {
                        if (picture != null) {
                                picture.delete();
                        }
                        iPacket.delete();
                        // ByteBufferUtil.destroy(data);
                }
                return null;
        }
}

private class decodeTask implements Runnable {

        private final Object msg;

        public decodeTask(Object msg) {
                super();
                this.msg = msg;
        }

        @Override
        public void run() {
                if (msg == null) {
                        return;
                }
                ChannelBuffer frameBuffer;
                if (frameDecoder != null) {
                        try {
                                frameBuffer = frameDecoder.decode((ChannelBuffer) msg);
                                if (frameBuffer == null) {
                                        return;
                                }
                        } catch (Exception e) {
                                return;
                        }

                } else {
                        frameBuffer = (ChannelBuffer) msg;
                }
```

```
                        int size = frameBuffer.readableBytes();
                        logger.info("decode the frame size :{}", size);
                        // start to decode
                        IBuffer iBuffer = IBuffer.make(null, size);
                        IPacket iPacket = IPacket.make(iBuffer);
                        iPacket.getByteBuffer().put(frameBuffer.toByteBuffer());
                        // decode the packet
                        if (!iPacket.isComplete()) {
                                return;
                        }

                        IVideoPicture picture = IVideoPicture.make(IPixelFormat.Type.YUV420P,
                                dimension.width, dimension.height);
                        try {
                                // decode the packet into the video picture
                                int postion = 0;
                                int packageSize = iPacket.getSize();
                                while (postion < packageSize) {
                                        postion += iStreamCoder.decodeVideo(picture, iPacket, postion);
                                        if (postion < 0) {
                                                throw new RuntimeException("error "
                                                        + " decoding video");
                                        }
                                        // if this is a complete picture, dispatch the picture
                                        if (picture.isComplete()) {
                                                IConverter converter = ConverterFactory.createConverter(type
                                                        .getDescriptor(), picture);
                                                BufferedImage image = converter.toImage(picture);
                                                // BufferedImage convertedImage = ImageUtils.convertToType(image,
                                                // BufferedImage.TYPE_3BYTE_BGR);
                                                // here ,put out the image
                                                if (streamFrameListener != null) {
                                                        streamFrameListener.onFrameReceived(image);
                                                }
                                                converter.delete();
                                        } else {
                                                picture.delete();
                                                iPacket.delete();
                                                return;
                                        }
                                        // clean the picture and reuse it
                                        picture.getByteBuffer().clear();
                                }
                        } finally {
                                if (picture != null) {
                                        picture.delete();
                                }
                                iPacket.delete();
                                // ByteBufferUtil.destroy(data);
                        }
                        return;
                }

        }

}
```

H264StreamEncoder class:

```
package conclaveclient.Handlers.utlity;
//Taken from SARXOS webcam example
import com.xuggle.xuggler.ICodec;
import com.xuggle.xuggler.IMetaData;
import com.xuggle.xuggler.IPacket;
import com.xuggle.xuggler.IPixelFormat.Type;
import com.xuggle.xuggler.IRational;
import com.xuggle.xuggler.IStreamCoder;
import com.xuggle.xuggler.IStreamCoder.Direction;
import com.xuggle.xuggler.IVideoPicture;
import com.xuggle.xuggler.video.ConverterFactory;
import com.xuggle.xuggler.video.IConverter;
import java.awt.Dimension;
import java.awt.image.BufferedImage;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import org.jboss.netty.buffer.BigEndianHeapChannelBuffer;
import org.jboss.netty.buffer.ChannelBuffer;
import org.jboss.netty.buffer.ChannelBuffers;
import org.jboss.netty.channel.Channel;
import org.jboss.netty.channel.ChannelHandlerContext;

import org.jboss.netty.handler.codec.oneone.OneToOneEncoder;

/**
 *
 * @author BradleyW
 */
public class H264StreamEncoder extends OneToOneEncoder{
	//protected final static Logger logger = LoggerFactory.getLogger(Logger.class);
	protected final IStreamCoder iStreamCoder = IStreamCoder.make(Direction.ENCODING, ICodec.ID.CODEC_ID_H264);
	protected final IPacket iPacket = IPacket.make();
	protected long startTime ;
	protected final Dimension dimension;
	protected final FrameEncoder frameEncoder;


	public H264StreamEncoder(Dimension dimension,boolean usingInternalFrameEncoder) {
		super();
		this.dimension = dimension;
		if (usingInternalFrameEncoder) {
			frameEncoder = new FrameEncoder(4);
		}else {
			frameEncoder = null;
		}
		initialize();
	}

	private void initialize(){
		//setup
		iStreamCoder.setNumPicturesInGroupOfPictures(25);

		iStreamCoder.setBitRate(200000);
		iStreamCoder.setBitRateTolerance(10000);
		iStreamCoder.setPixelType(Type.YUV420P);
```

```
                iStreamCoder.setHeight(dimension.height);
                iStreamCoder.setWidth(dimension.width);
                iStreamCoder.setFlag(IStreamCoder.Flags.FLAG_QSCALE, true);
                iStreamCoder.setGlobalQuality(0);
                //rate
                IRational rate = IRational.make(25, 1);
                iStreamCoder.setFrameRate(rate);
                //time base
                //iStreamCoder.setAutomaticallyStampPacketsForStream(true);
                iStreamCoder.setTimeBase(IRational.make(rate.getDenominator(),rate.getNumerator()));
                IMetaData codecOptions = IMetaData.make();
                codecOptions.setValue("tune", "zerolatency");// equals -tune zerolatency in ffmpeg
                //open it
                int revl = iStreamCoder.open(codecOptions, null);
                if (revl < 0) {
                        throw new RuntimeException("could not open the coder");
                }
}



@Override
protected Object encode(ChannelHandlerContext ctx, Channel channel,
                        Object msg) throws Exception {
                return encode(msg);
}

public Object encode(Object msg) throws Exception {
                if (msg == null || msg instanceof BigEndianHeapChannelBuffer) {
                        return null;
                }
                if (!(msg instanceof BufferedImage)) {
System.out.println("Class: " + msg.getClass());
                        throw new IllegalArgumentException("your need to pass into an bufferedimage");
                }
                //logger.info("encode the frame");
                BufferedImage bufferedImage = (BufferedImage)msg;
                //here is the encode
                //convert the image
                BufferedImage convetedImage = ImageUtils.convertToType(bufferedImage, BufferedImage.TYPE_3BYTE_BGR);
                IConverter converter = ConverterFactory.createConverter(convetedImage, Type.YUV420P);
                //to frame
                long now = System.currentTimeMillis();
                if (startTime == 0) {
                                        startTime = now;
                }
                IVideoPicture pFrame = converter.toPicture(convetedImage, (now - startTime)*1000);
                //pFrame.setQuality(0);
                iStreamCoder.encodeVideo(iPacket, pFrame, 0) ;
                //free the MEM
                pFrame.delete();
                converter.delete();
                //write to the container
                if (iPacket.isComplete()) {

                        //iPacket.delete();
```

```
                        //here we send the package to the remote peer
                        try{
                                ByteBuffer byteBuffer = iPacket.getByteBuffer();
                                if (iPacket.isKeyPacket()) {
                                        //logger.info("key frame");
                                }
                                ChannelBuffer channelBuffe =
ChannelBuffers.copiedBuffer(byteBuffer.order(ByteOrder.BIG_ENDIAN));
                                if (frameEncoder != null) {
                                        return frameEncoder.encode(channelBuffe);
                                }
                                return channelBuffe;

                        }finally{
                                iPacket.reset();
                        }
                }else{
                        return null;
                }
        }

}
```

ImageUtils class:

```
package conclaveclient.Handlers.utlity;
//Taken from SARXOS webcam example
import java.awt.image.BufferedImage;


public class ImageUtils {

        public static BufferedImage convertToType(BufferedImage sourceImage,
           int targetType)
        {
          BufferedImage image;

          // if the source image is already the target type, return the source image

          if (sourceImage.getType() == targetType)
            image = sourceImage;

          // otherwise create a new image of the target type and draw the new
          // image

          else
          {
            image = new BufferedImage(sourceImage.getWidth(),
               sourceImage.getHeight(), targetType);
            image.getGraphics().drawImage(sourceImage, 0, 0, null);
          }

          return image;
        }
}
```

PacketUtil class:

```
package conclaveclient.Handlers.utlity;

import conclaveclient.security.Encryptor;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author BradleyW
 */
public class PacketUtil {

   private Socket outSocket;
   private InetAddress ip;
   private int port;
   private final int timeOutPeriod = 2000;
   private String secKey;
   private String skeyUser;

   public PacketUtil(InetAddress ip, int port, String skey, String skeyUser) throws IOException {
      this.ip = ip;
      this.port = port;
      outSocket = new Socket(ip, port);
      outSocket.setSoTimeout(timeOutPeriod);
      this.secKey = skey;
      this.skeyUser = skeyUser;
   }

   public void refreshSocket() {
      try {
         outSocket = new Socket(ip, port);
      } catch (IOException e) {
      }
   }

   public void setConnectionDetails(InetAddress ip, int port) {
      this.ip = ip;
      this.port = port;
      refreshSocket();
   }

   public void sendPacketRequest(String request) throws UnknownHostException {
      try {
         String encMessage = Encryptor.encrypt(request, secKey);
         String writeMsg = skeyUser + encMessage
               + "\n";
         BufferedOutputStream bos = new BufferedOutputStream(outSocket.getOutputStream());
         OutputStreamWriter osw = new OutputStreamWriter(bos, "UTF-8");
```

```
            System.out.println("Sending: " + writeMsg);
            osw.write(writeMsg);
            osw.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public String readStream() {
        String entireRequest = "";
        try {
            InputStreamReader insr = new InputStreamReader(outSocket.getInputStream());
            BufferedReader br = new BufferedReader(insr);
            String nextLine = "";
            int timeOut = 0;
            while (timeOut < timeOutPeriod) {
                if ((nextLine = br.readLine()) != null) {
                    entireRequest = entireRequest + nextLine;
                    if (!br.ready()) {
                        break;
                    } else {
                        entireRequest = entireRequest + "\n";
                    }
                } else {
                    timeOut++;
                    Thread.sleep(1);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException ex) {
            Logger.getLogger(PacketUtil.class.getName()).log(Level.SEVERE, null, ex);
        }
        String res;
        if (!entireRequest.contains("403"))
        {
            res = Encryptor.decrypt(entireRequest, secKey);
            System.out.println("Recieving: " + res);
        } else {
            res = entireRequest;
        }
        return res;
    }

    public void close() {
        try {
            outSocket.close();
        } catch (IOException e) {

        }
    }
}
```

RMISecurityPolicyLoader class:

```
package conclaveclient.rmi;
import java.net.URL;
import java.rmi.RMISecurityManager;
```

```
/**
 *
 * Taken from DHB tutorial
 */
public class RMISecurityPolicyLoader {

   private static boolean loaded = false;
   public static void LoadDefaultPolicy()
   {
      LoadPolicy("RMISecurity.policy");
   }
   public static void LoadPolicy(String policy) {
      if (loaded) {    return;    }
      loaded = true;
      ClassLoader cl = RMISecurityPolicyLoader.class.getClassLoader();
      URL url = cl.getResource(policy) ;
      if (url == null) {
         System.out.println("Policy not found null (Have you included it in the jar file? (Don't forget to clean and build))");
      } else {
         System.out.println("Policy found");
      }
      System.out.println(url.toString());
      System.setProperty("java.security.policy", url.toString());

      if (System.getSecurityManager() == null) {
         System.out.println("Creating security policy");
         System.setSecurityManager(new RMISecurityManager());
      }

   }

   private RMISecurityPolicyLoader() {
   }
}
```

Encryptor class:

```
package conclaveclient.security;

import java.nio.charset.Charset;
import java.security.GeneralSecurityException;
import java.security.InvalidKeyException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import javax.xml.bind.DatatypeConverter;

/**
 *
 * @author BradleyW
 */
public class Encryptor {

   public Encryptor() {
```

```
}

public static String encrypt(final String plainMessage,final String symKeyHex) {
   final byte[] symKeyData = DatatypeConverter.parseHexBinary(symKeyHex);

   final byte[] encodedMessage = plainMessage.getBytes(Charset
         .forName("UTF-8"));
   try {
      final Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
      final int blockSize = cipher.getBlockSize();

      // create the key
      final SecretKeySpec symKey = new SecretKeySpec(symKeyData, "AES");

      // generate random IV using block size (possibly create a method for
      // this)
      final byte[] ivData = new byte[blockSize];
      final SecureRandom rnd = SecureRandom.getInstance("SHA1PRNG");
      rnd.nextBytes(ivData);
      final IvParameterSpec iv = new IvParameterSpec(ivData);

      cipher.init(Cipher.ENCRYPT_MODE, symKey, iv);

      final byte[] encryptedMessage = cipher.doFinal(encodedMessage);

      // concatenate IV and encrypted message
      final byte[] ivAndEncryptedMessage = new byte[ivData.length
            + encryptedMessage.length];
      System.arraycopy(ivData, 0, ivAndEncryptedMessage, 0, blockSize);
      System.arraycopy(encryptedMessage, 0, ivAndEncryptedMessage,
            blockSize, encryptedMessage.length);

      final String ivAndEncryptedMessageBase64 = DatatypeConverter
            .printBase64Binary(ivAndEncryptedMessage);

      return ivAndEncryptedMessageBase64;
   } catch (InvalidKeyException e) {
      throw new IllegalArgumentException(
            "key argument does not contain a valid AES key");
   } catch (GeneralSecurityException e) {
      throw new IllegalStateException(
            "Unexpected exception during encryption", e);
   }
}

public static String decrypt(final String ivAndEncryptedMessageBase64,
      final String symKeyHex) {
   final byte[] symKeyData = DatatypeConverter.parseHexBinary(symKeyHex);

   final byte[] ivAndEncryptedMessage = DatatypeConverter
         .parseBase64Binary(ivAndEncryptedMessageBase64);
   try {
      final Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
      final int blockSize = cipher.getBlockSize();

      // create the key
```

```
            final SecretKeySpec symKey = new SecretKeySpec(symKeyData, "AES");

            // retrieve random IV from start of the received message
            final byte[] ivData = new byte[blockSize];
            System.arraycopy(ivAndEncryptedMessage, 0, ivData, 0, blockSize);
            final IvParameterSpec iv = new IvParameterSpec(ivData);

            // retrieve the encrypted message itself
            final byte[] encryptedMessage = new byte[ivAndEncryptedMessage.length
                  - blockSize];
            System.arraycopy(ivAndEncryptedMessage, blockSize,
                  encryptedMessage, 0, encryptedMessage.length);

            cipher.init(Cipher.DECRYPT_MODE, symKey, iv);

            final byte[] encodedMessage = cipher.doFinal(encryptedMessage);

            // concatenate IV and encrypted message
            final String message = new String(encodedMessage,
                  Charset.forName("UTF-8"));

            return message;
         } catch (InvalidKeyException e) {
            throw new IllegalArgumentException(
                  "key argument does not contain a valid AES key");
         } catch (GeneralSecurityException e) {
            throw new IllegalStateException(
                  "Unexpected exception during decryption", e);
         }
      }

      public static String hashPassword(String password, byte[] salt) {
         String hashedString = null;
         try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            md.update(salt);
            byte[] hashbytes = md.digest(password.getBytes());
            StringBuilder sb = new StringBuilder();
            for (int i = 0; i < hashbytes.length; i++) {
               sb.append(Integer.toString((hashbytes[i] & 0xff) + 0x100, 16).substring(1));
            }
            hashedString = sb.toString();
         } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
         }
         return hashedString;
      }
   }
```

Testing Classes:

AdminInteractionTests class:

```
package conclaveclient;

import conclaveinterfaces.IAdminInterface;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
```

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author BradleyW
 */
public class AdminInteractionTests {

   private static IAdminInterface ui;
   private static LoginController controller;

   //Static Secret Key Variables for AES encryption setup.
   private static final String secKey = "B55E4C33045B62AC907529233ADAAD6C";
   private static final String secKeyUser = "00036";

   public AdminInteractionTests() {
   }

   @BeforeClass
   public static void setUpClass() {
      controller = new LoginController(secKey, secKeyUser);
      controller.connect("192.168.0.15", "20003");
      controller.createAccount("AdminAdam", "password123");
      try {
         controller.login("AdminAdam", "password123");
         Registry registry = LocateRegistry.getRegistry("192.168.0.15", 9807);
         ui = (IAdminInterface) registry.lookup("AdminAdam");
      } catch (RemoteException ex) {
         Logger.getLogger(AdminInteractionTests.class.getName()).log(Level.SEVERE, null, ex);
      } catch (NotBoundException ex) {
         Logger.getLogger(AdminInteractionTests.class.getName()).log(Level.SEVERE, null, ex);
      }
   }

   @AfterClass
   public static void tearDownClass() {
   }

   @Before
   public void setUp() {
   }

   @After
   public void tearDown() throws RemoteException {
      if (ui.inRoom()) {
         ui.leaveRoom();
```

```
    }
  }

  @Test
  public void adminInteractionTest1() {
    try {
      ui.addRoom("newRoom", 1);
      boolean successfull = ui.joinRoom("newRoom");
      assertEquals(successfull, true);
    } catch (RemoteException ex) {
      Logger.getLogger(AdminInteractionTests.class.getName()).log(Level.SEVERE, null, ex);
    }
  }

  @Test
  public void adminInteractionTest2() {
    try {
      ui.addRoom("newRoom", 1);
      List<String> names = ui.getRoomNames();
      for (String name : names) {
        System.out.println(name);
      }
      boolean successfull = false;
      assertEquals(successfull, true);
    } catch (RemoteException ex) {
      Logger.getLogger(AdminInteractionTests.class.getName()).log(Level.SEVERE, null, ex);
    }
  }

}
```

LoginControllerConnectTests:

```
package conclaveclient;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author BradleyW
 */
public class LoginControllerConnectTests {

  //Static Secret Key Variables for AES encryption setup.
  private static final String secKey = "B55E4C33045B62AC907529233ADAAD6C";
  private static final String secKeyUser = "00036";

  public LoginControllerConnectTests() {
  }

  @Before
  public void setUp() {

  }
```

```
@After
public void tearDown() {
}

/**
 * Test of connect method, of class LoginController.
 */
@Test
public void testConnect1() {
   System.out.println("connect");
   String iip = "192.168.0.15";
   String iport = "20003";
   LoginController instance = new LoginController(secKey, secKeyUser);
   String expResult = "Connected to server at: /" + iip +":"+ iport;
   String result = instance.connect(iip, iport);
   assertEquals(expResult, result);
}

@Test
public void testConnect2() {
   System.out.println("connect");
   String iip = "192.168.0.8";
   String iport = "20003";
   LoginController instance = new LoginController(secKey, secKeyUser);
   String expResult = "Cannot find a conclave server on that IP or Port";
   String result = instance.connect(iip, iport);
   assertEquals(expResult, result);
}
@Test
public void testConnect3() {
   System.out.println("connect");
   String iip = "192.168.0.3";
   String iport = "20004";
   LoginController instance = new LoginController(secKey, secKeyUser);
   String expResult = "Cannot find a conclave server on that IP or Port";
   String result = instance.connect(iip, iport);
   assertEquals(expResult, result);
}
@Test
public void testConnect4() {
   System.out.println("connect");
   String iip = "192.168.0.8";
   String iport = "ACDE";
   LoginController instance = new LoginController(secKey, secKeyUser);
   String expResult = "Bad format of IP/Port";
   String result = instance.connect(iip, iport);
   assertEquals(expResult, result);
}
@Test
public void testConnect5() {
   System.out.println("connect");
   String iip = "ACDE";
   String iport = "20003";
   LoginController instance = new LoginController(secKey, secKeyUser);
   String expResult = "Bad format of IP/Port";
   String result = instance.connect(iip, iport);
```

```
    assertEquals(expResult, result);
  }
  @Test
  public void testConnect6() {
    System.out.println("connect");
    String iip = "ACDE";
    String iport = "ACDE";
    LoginController instance = new LoginController(secKey, secKeyUser);
    String expResult = "Bad format of IP/Port";
    String result = instance.connect(iip, iport);
    assertEquals(expResult, result);
  }
  @Test
  public void testConnect7() {
    System.out.println("connect");
    String iip = "";
    String iport = "20003";
    LoginController instance = new LoginController(secKey, secKeyUser);
    String expResult = "Bad format of IP/Port";
    String result = instance.connect(iip, iport);
    assertEquals(expResult, result);
  }@Test
  public void testConnect8() {
    System.out.println("connect");
    String iip = "192.168.0.3";
    String iport = "";
    LoginController instance = new LoginController(secKey, secKeyUser);
    String expResult = "Bad format of IP/Port";
    String result = instance.connect(iip, iport);
    assertEquals(expResult, result);
  }
  @Test
  public void testConnect9() {
    System.out.println("connect");
    String iip = "";
    String iport = "";
    LoginController instance = new LoginController(secKey, secKeyUser);
    String expResult = "Bad format of IP/Port";
    String result = instance.connect(iip, iport);
    assertEquals(expResult, result);
  }

}
```

LoginControllerCreateAccountTests:

```
package conclaveclient;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author BradleyW
```

```
 */
public class LoginControllerCreateAccountTest {

   private static LoginController controller;
 //Static Secret Key Variables for AES encryption setup.
   private static final String secKey = "B55E4C33045B62AC907529233ADAAD6C";
   private static final String secKeyUser = "00036";

   public LoginControllerCreateAccountTest() {
   }

   @BeforeClass
   public static void setUpClass() {
      controller = new LoginController(secKey, secKeyUser);
      controller.connect("192.168.0.15", "20003");
   }

   @AfterClass
   public static void tearDownClass() {
   }

   @Before
   public void setUp() {
   }

   @After
   public void tearDown() {
   }

   @Test
   public void testCreateAccount1() {
      String username = "TestAccount";
      String password = "password1421";
      String expResult = "100 REQUEST VALID Account creation success";
      String result = controller.createAccount(username, password);
      assertEquals(expResult, result);
   }

   @Test
   public void testCreateAccount2() {
      String username = "923464252sf1f";
      String password = "3515sf1fe1f1";
      String expResult = "100 REQUEST VALID Account creation success";
      String result = controller.createAccount(username, password);
      assertEquals(expResult, result);
   }

   @Test
   public void testCreateAccount3() {
      String username = "ADdsa&£$!&*$";
      String password = "afga$faf3a";
      String expResult = "100 REQUEST VALID Account creation success";
      String result = controller.createAccount(username, password);
      assertEquals(expResult, result);
   }
```

```
@Test
public void testCreateAccount4() {
   String username = "UserAdam";
   String password = "password123";
   String expResult = "202 ACCOUNT CREATE UNSUCCESSFUL That user already exists";
   String result = controller.createAccount(username, password);
   assertEquals(expResult, result);
}

@Test
public void testCreateAccount5() {
   String username = "123456";
   String password = "123";
   String expResult = "400 REQUEST SYNTAX ERROR Username and Password must be atleast 5 characters long";
   String result = controller.createAccount(username, password);
   assertEquals(expResult, result);
}

@Test
public void testCreateAccount6() {
   String username = "123";
   String password = "123456";
   String expResult = "400 REQUEST SYNTAX ERROR Username and Password must be atleast 5 characters long";
   String result = controller.createAccount(username, password);
   assertEquals(expResult, result);
}

@Test
public void testCreateAccount7() {
   String username = "1234567890123456";
   String password = "123456";
   String expResult = "400 REQUEST SYNTAX ERROR Username and Password must be less than 15 characters long";
   String result = controller.createAccount(username, password);
   assertEquals(expResult, result);
}

@Test
public void testCreateAccount8() {
   String username = "thixists2";
   String password = "1234567890123456";
   String expResult = "400 REQUEST SYNTAX ERROR Username and Password must be less than 15 characters long";
   String result = controller.createAccount(username, password);
   assertEquals(expResult, result);
}

@Test
public void testCreateAccount9() {
   String username = "";
   String password = "123456789012345";
   String expResult = "400 REQUEST SYNTAX ERROR No username or password given";
   String result = controller.createAccount(username, password);
   assertEquals(expResult, result);
}

@Test
public void testCreateAccount10() {
```

```
        String username = "thixists";
        String password = "";
        String expResult = "400 REQUEST SYNTAX ERROR No username or password given";
        String result = controller.createAccount(username, password);
        assertEquals(expResult, result);
    }

    @Test
    public void testCreateAccount11() {
        String username = "";
        String password = "";
        String expResult = "400 REQUEST SYNTAX ERROR No username or password given";
        String result = controller.createAccount(username, password);
        assertEquals(expResult, result);
    }
}
```

LoginControllerKeyTests class:

```
package conclaveclient;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author BradleyW
 */
public class LoginControllerKeyTests {

    //Static Secret Key Variables for AES encryption setup.
    private static final String iip = "192.168.0.15";
    private static final String iport = "20003";
    public LoginControllerKeyTests() {
    }

    @Before
    public void setUp() {

    }

    @After
    public void tearDown() {
    }

    /**
     * Test of connect method, of class LoginController.
     */
    @Test
    public void testKey1() {
        String secKey = "B55E4C33045B62AC907529233ADAAD6C";
        String secKeyUsers = "00001";
        LoginController instance = new LoginController(secKey, secKeyUsers);
        instance.connect(iip, iport);
        String expResult = "403 NOT PERMITTED Invalid key given";
        String result = instance.connect(iip, iport);
```

```
    assertEquals(expResult, result);
  }

  @Test
  public void testKey2() {
    String secKey = "B55E4C33045B62AC907529233ADAAD6C";
    String secKeyUsers = "00036";
    LoginController instance = new LoginController(secKey, secKeyUsers);
    instance.connect(iip, iport);
    String expResult = "Connected to server at: /" + iip +":"+ iport;
    String result = instance.connect(iip, iport);
    assertEquals(expResult, result);
  }

}
```

LoginControllerLoginTests class:

```
package conclaveclient;

import java.rmi.RemoteException;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author BradleyW
 */
public class LoginControllerLoginTests {

  private static LoginController controller;
   //Static Secret Key Variables for AES encryption setup.
  private static final String secKey = "B55E4C33045B62AC907529233ADAAD6C";
  private static final String secKeyUser = "00036";


  public LoginControllerLoginTests() {
  }

  @BeforeClass
  public static void setUpClass() {
    controller = new LoginController(secKey, secKeyUser);
    controller.connect("192.168.0.15","20003");
    controller.createAccount("UserAdam", "password123");
    controller.createAccount("AdminsAdam", "adminpassw");
    controller.createAccount("KandyKane", "password124");
  }

  @AfterClass
  public static void tearDownClass() {
  }
```

```
@Before
public void setUp() {
   controller.connect("192.168.0.15","20003");
   if (!controller.isConnected())
   {
      System.out.println("Cannot connect to the conclave server");
   }
}

@After
public void tearDown() {
}

@Test
public void testLogin1()
{
   try {
      String username = "UserAdam";
      String password = "password123";
      String expResult = "100 REQUEST VALID You have logged in as a User: " + username;
      String result = controller.login(username, password);
      assertEquals(expResult, result);
   } catch (RemoteException ex) {
      Logger.getLogger(LoginControllerLoginTests.class.getName()).log(Level.SEVERE, null, ex);
   }
}

@Test
public void testLogin2()
{
   try {
      String username = "AdminsAdam";
      String password = "adminpassw";
      String expResult = "100 REQUEST VALID You have logged in as an Admin: " + username;
      String result = controller.login(username, password);
      assertEquals(expResult, result);
   } catch (RemoteException ex) {
      Logger.getLogger(LoginControllerLoginTests.class.getName()).log(Level.SEVERE, null, ex);
   }
}

@Test
public void testLogin3()
{
   try {
      String username = "UserAdam";
      String password = "password123";
      String expResult = "423 USER ALREADY LOGGED IN That user is already logged in";
      String result = controller.login(username, password);
      assertEquals(expResult, result);
   } catch (RemoteException ex) {
      Logger.getLogger(LoginControllerLoginTests.class.getName()).log(Level.SEVERE, null, ex);
   }
}

@Test
```

```
public void testLogin4()
{
   try {
      String username = "KandyKane";
      String password = "wrongpassword";
      String expResult = "401 INCORRECT LOGIN DETAILS Incorrect username/password";
      String result = controller.login(username, password);
      assertEquals(expResult, result);
   } catch (RemoteException ex) {
      Logger.getLogger(LoginControllerLoginTests.class.getName()).log(Level.SEVERE, null, ex);
   }
}

@Test
public void testLogin5()
{
   try {
      String username = "-------";
      String password = "Periapsis";
      String expResult = "404 CANNOT LOCATE RESOURCE A user by that username could not be found";
      String result = controller.login(username, password);
      assertEquals(expResult, result);
   } catch (RemoteException ex) {
      Logger.getLogger(LoginControllerLoginTests.class.getName()).log(Level.SEVERE, null, ex);
   }
}

@Test
public void testLogin6()
{
   try {
      String username = "-------";
      String password = "Periapsis";
      String expResult = "404 CANNOT LOCATE RESOURCE A user by that username could not be found";
      String result = controller.login(username, password);
      assertEquals(expResult, result);
   } catch (RemoteException ex) {
      Logger.getLogger(LoginControllerLoginTests.class.getName()).log(Level.SEVERE, null, ex);
   }
}

@Test
public void testLogin7()
{
   try {
      String username = "";
      String password = "";
      String expResult = "400 REQUEST SYNTAX ERROR No Username/Password given";
      String result = controller.login(username, password);
      assertEquals(expResult, result);
   } catch (RemoteException ex) {
      Logger.getLogger(LoginControllerLoginTests.class.getName()).log(Level.SEVERE, null, ex);
   }
}

@Test
```

```
    public void testLogin8()
    {
        try {
            String username = "LOALOSLO";
            String password = "";
            String expResult = "400 REQUEST SYNTAX ERROR No Username/Password given";
            String result = controller.login(username, password);
            assertEquals(expResult, result);
        } catch (RemoteException ex) {
            Logger.getLogger(LoginControllerLoginTests.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    @Test
    public void testLogin9()
    {
        try {
            String username = "";
            String password = "LOALOSLO";
            String expResult = "400 REQUEST SYNTAX ERROR No Username/Password given";
            String result = controller.login(username, password);
            assertEquals(expResult, result);
        } catch (RemoteException ex) {
            Logger.getLogger(LoginControllerLoginTests.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    @Test
    public void testLogin10()
    {
        try {
            String username = "";
            String password = "LOALOSLO";
            String expResult = "400 REQUEST SYNTAX ERROR No Username/Password given";
            String result = controller.login(username, password);
            assertEquals(expResult, result);
        } catch (RemoteException ex) {
            Logger.getLogger(LoginControllerLoginTests.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

UserInteractionTests class:

```
package conclaveclient;

import conclaveinterfaces.IUserInterface;
import java.rmi.AccessException;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import model.Message;
import org.junit.After;
import org.junit.AfterClass;
```

```
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author BradleyW
 */
public class UserInteractionTests {

   private static IUserInterface ui;
   private static LoginController controller;
 //Static Secret Key Variables for AES encryption setup.
   private static final String secKey = "B55E4C33045B62AC907529233ADAAD6C";
   private static final String secKeyUser = "00036";

   public UserInteractionTests() {
   }

   @BeforeClass
   public static void setUpClass() {
      controller = new LoginController(secKey, secKeyUser);
      controller.connect("192.168.0.15", "20003");
      controller.createAccount("UserAdam", "password123");
      try {
         controller.login("UserAdam", "password123");
         Registry registry = LocateRegistry.getRegistry("192.168.0.15", 9807);
         ui = (IUserInterface) registry.lookup("UserAdam");
      } catch (RemoteException ex) {
         Logger.getLogger(UserInteractionTests.class.getName()).log(Level.SEVERE, null, ex);
      } catch (NotBoundException ex) {
         Logger.getLogger(UserInteractionTests.class.getName()).log(Level.SEVERE, null, ex);
      }
   }

   @AfterClass
   public static void tearDownClass() {
   }

   @Before
   public void setUp() {
   }

   @After
   public void tearDown() throws RemoteException {
      if (ui.inRoom())
      {
         ui.leaveRoom();
      }
   }

   @Test
   public void joinRoomTest1() {
      try {
         ui.joinRoom("Atrium");
```

```
        boolean successfull = ui.inRoom();
        assertEquals(successfull, true);
    } catch (RemoteException ex) {
        Logger.getLogger(UserInteractionTests.class.getName()).log(Level.SEVERE, null, ex);
    }
}

@Test
public void joinRoomTest2() {
    try {
        ui.joinRoom("Atrium");
        ui.leaveRoom();
        boolean successfull = ui.inRoom();
        assertEquals(successfull, false);
    } catch (RemoteException ex) {
        Logger.getLogger(UserInteractionTests.class.getName()).log(Level.SEVERE, null, ex);
    }
}

@Test
public void joinRoomTest3() {
    try {
        System.out.println(ui.joinRoom("TestingRoom", "test123"));
        boolean successfull = ui.inRoom();
        assertEquals(successfull, true);
    } catch (RemoteException ex) {
        System.out.println("error");
        Logger.getLogger(UserInteractionTests.class.getName()).log(Level.SEVERE, null, ex);
    }
}

@Test
public void joinRoomTest4() {
    try {
        ui.joinRoom("TestingRoom", "wrongpass");
        boolean successful = ui.inRoom();
        assertEquals(successful, false);
    } catch (RemoteException ex) {
        Logger.getLogger(UserInteractionTests.class.getName()).log(Level.SEVERE, null, ex);
    }
}

@Test
public void joinRoomTest5() {
    try {
        ui.joinRoom("Locked Conference", "test123");
        ui.leaveRoom();
        boolean successfull = ui.inRoom();
        assertEquals(successfull, false);
    } catch (RemoteException ex) {
        Logger.getLogger(UserInteractionTests.class.getName()).log(Level.SEVERE, null, ex);
    }
}

@Test
public void joinRoomTest6() {
```

```
    try {
      ui.joinRoom("Closed Room");
      boolean successfull = ui.inRoom();
      assertEquals(successfull, false);
    } catch (RemoteException ex) {
      Logger.getLogger(UserInteractionTests.class.getName()).log(Level.SEVERE, null, ex);
    }
  }

  @Test
  public void joinRoomTest7() {
    try {
      boolean successful = true;
      ui.postMessage("Testing Message");
      List<Message> chatUpdate = ui.getChatlogUpdates(0);
      for (Message msg : chatUpdate)
      {
        if (msg.msgDisplay().contains("Testing Message"))
        {
          successful = false;
        }
      }
      assertEquals(successful, true);
    } catch (RemoteException ex) {
      Logger.getLogger(UserInteractionTests.class.getName()).log(Level.SEVERE, null, ex);
    }
  }

  @Test
  public void joinRoomTest8() {
    try {
      boolean successful = false;
      ui.joinRoom("Atrium");
      ui.postMessage("Testing Message");
      List<Message> chatUpdate = ui.getChatlogUpdates(0);
      for (Message msg : chatUpdate)
      {
        if (msg.msgDisplay().contains("Testing Message"))
        {
          successful = true;
        }
      }
      assertEquals(successful, true);
    } catch (RemoteException ex) {
      Logger.getLogger(UserInteractionTests.class.getName()).log(Level.SEVERE, null, ex);
    }
  }

}
```