Intro to application:

The system is a client & server chat application where multiple clients can:

- Join, create, and leave chat channels.
- The server manages the chat channels and handles client connections.
- The client application provides a user interface for interacting with the chat system.

Design:

- Scalability: The server should handle multiple client connections simultaneously.
- Security: Messages are encrypted using AES encryption.
- Usability: The client UI should be intuitive and responsive.

System Overview:

The system follows a client*server architecture. The main components are:

- Server: Manages chat channels and client connections.
- Client: Provides a user interface for users to interact with the chat system.
- Communication: Uses sockets for communication between the client and server.
- Messages: classes sent between Client & Server for easy communication.

Important Classes for functionality:

Class: Server

Responsibilities:

- Listen for incoming client connections.
- Manage chat channels.
- Handle client requests.

Significant methods:

- startListening(): Starts the server to listen for client connections.
- stop(): Stops the server.
- createChannel(String channelName, String password): Creates a new chat channel.
- getChannel(String channelName): Retrieves a chat channel by name.

Important Attributes:

- Map<String, ChatChannel>: map of active chat channels.
- boolean isRunning: boolean for if server is active.

Class: ClientHandler

Responsibilities:

- Handle communication with a single client.
- Process client messages.

Significant methods:

- run(): Listens for incoming messages from the client.
- joinChannel(String channelName, String password): Joins the client to a chat channel.
- leaveChannel(String channelName): Removes the client from a chat channel.

- sendMessage(Message message): Sends a message to the client.

Class: Client
Responsibilities:
- Facade for managing client side operations.

Significant methods:
- createChannel(String channelName, String password): Sends a request to create a new channel.
- joinChannel(String channelName, String password): Sends a request to join a channel.
- leaveChannel(): Sends a request to leave the current channel.
- sendMessage(String message): Sends a message to the current channel.

Class: ClientCommunicationManager
Responsibilities:
- Handle communication between the client and server.

Significant methods:
- run(): Continuously reads messages from the server.
- sendMessageToServer(ServerMessage message): Sends a message to the server.

Class: UIController
Responsibilities:
- Handle user interactions.
- Update the UI based on server responses.

Significant methods:
- update(UIMessage message): Updates the UI based on the received message.
- createChannel(): Handles the creation of a new channel.
- joinChannel(): Handles joining an existing channel.
- showChatArea(): Displays the chat area in the UI.

Class: MessageVisitorUI
Responsibilities:
- Process UI messages.

Significant methods:
- handle(DisplayError e): Handles error messages.
- handle(DisplayMessage m): Handles display messages.
- handle(UpdateChannels u): Handles channel updates.

Class: EncryptionLayer
Responsibilities:
- Encrypt and decrypt messages using AES.

Significant methods:
- generateKey(): Generates a new AES key.
- encrypt(Object encryptedData, SecretKey key): Encrypts data.
- decrypt(String encryptedData, SecretKey key): Decrypts data.

Class: StandardView

Responsibilities:

- Display the main user interface
- Handle user interactions

Significant methods:

- startArea(): initializes the area of the view.
- showChatArea(): Displays the chat area.
- appendChatText(String txt): appends text to the chat area.
- add*ButtonListener(ActionsListener listener): Adds buttons for our user interface like leave/join/create channel.

Summary of how system works/communicates:

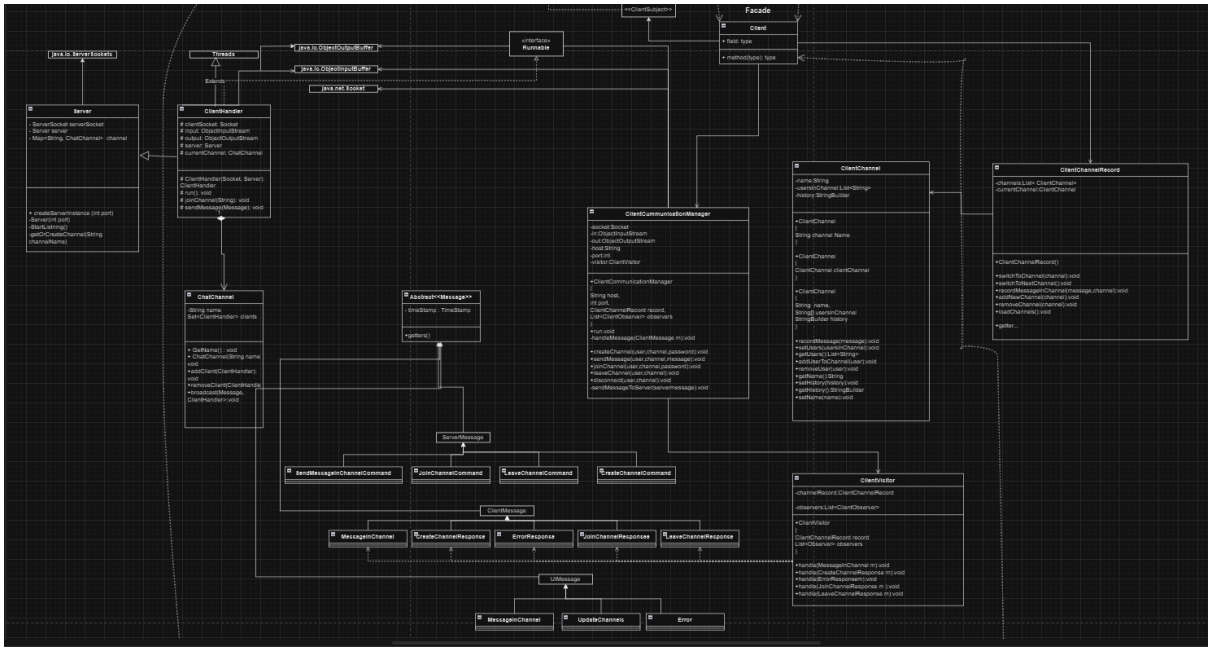
- Our program consists of Server & Client & UI, When the user interacts with our interface for ex presses buttons or sends a message. The UIController calls on our Client methods which in return passes it to the server through the socket with our Message class. If the user for example wants to join a channel, our UI calls on Client method which sends a JoinChannelCommand class to the server containing channel name and password, server then receives these messages in our ClientHandler, where we pattern match on the class type and handle it. When the ClientHandler has completed/validated the request it sends back a response message to the client or an error message containing a String of what went wrong which we print to the user.

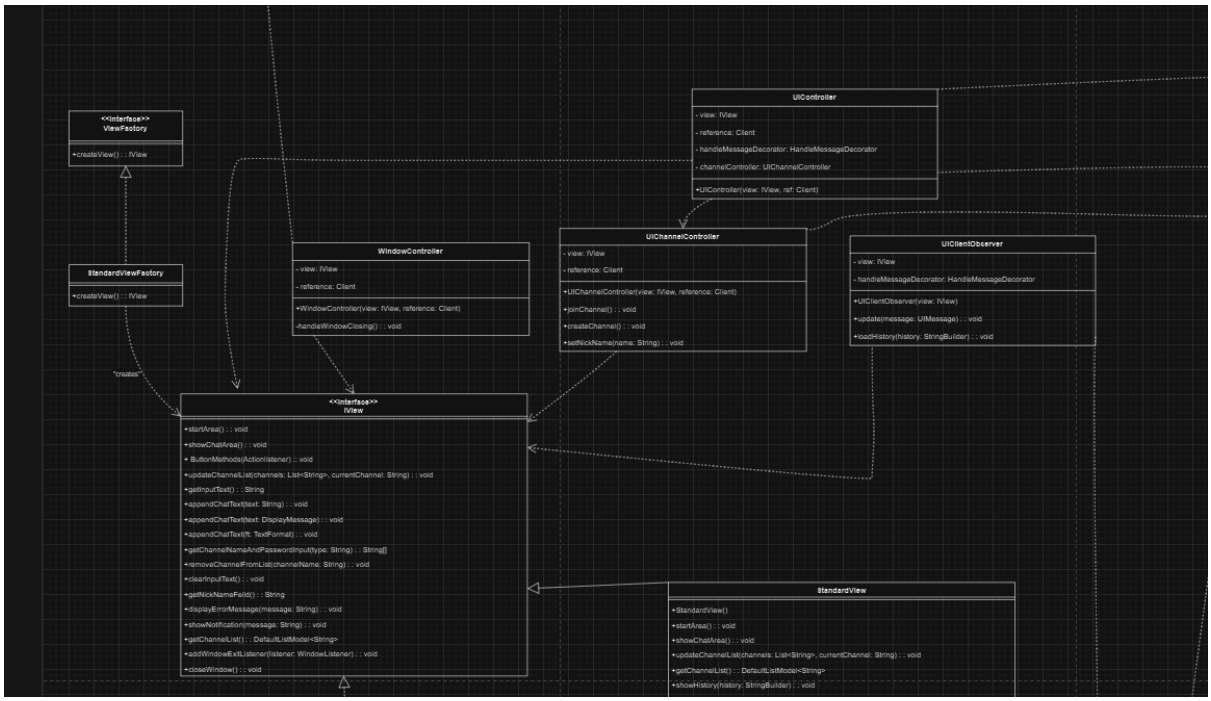
Design Patterns Implemented:

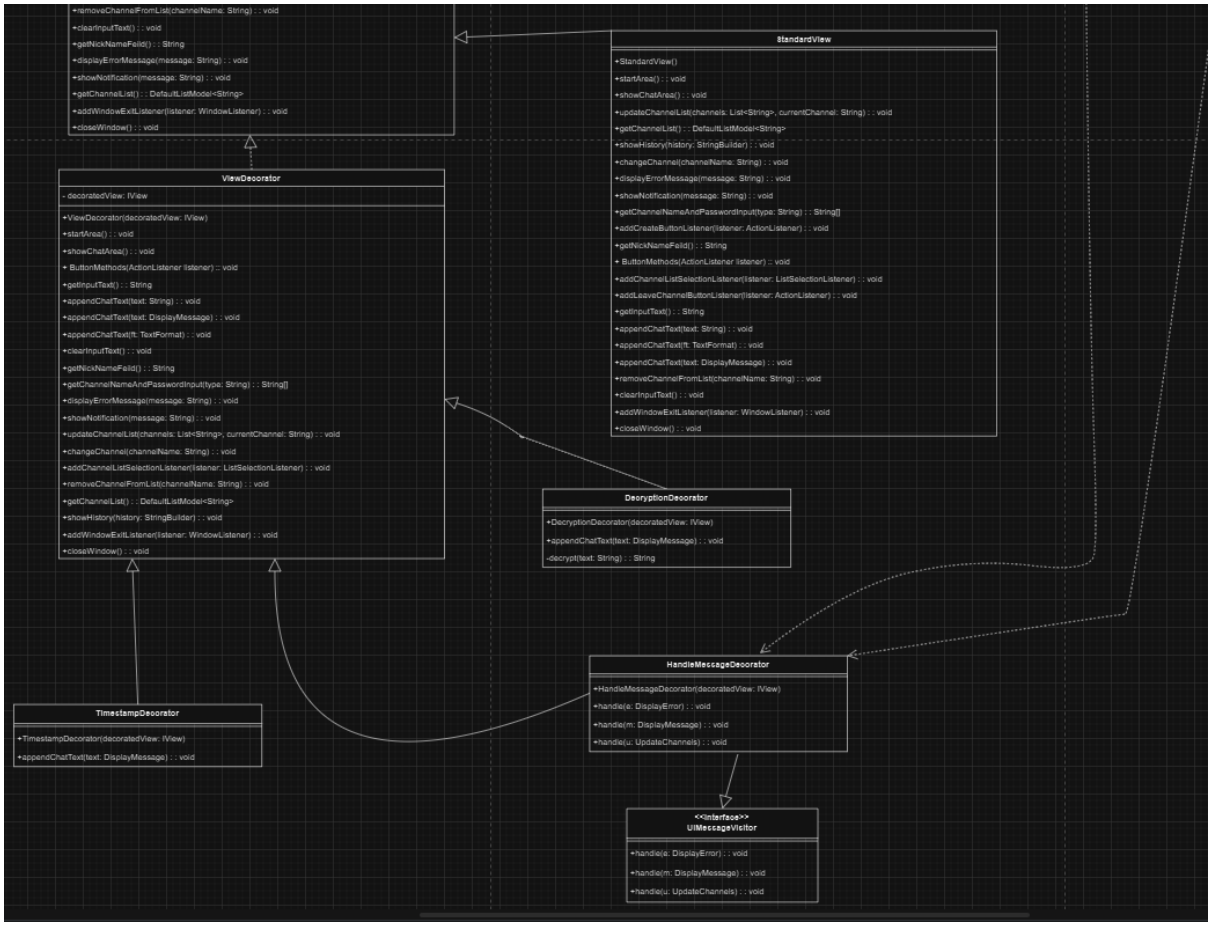
- For our messages we've implemented a **visitor pattern** in order to pattern match on different types of messages. This way we can easily handle the different types of messages and separate the server & client messages. Also easily scalable when implementing new features.
- For the View module we use **Decorator & factory pattern**, the purpose of the decorator pattern is to add functionality to objects dynamically (ViewDecorator & its subclasses wrap an IView object to add certain functionality). As well as **observer pattern** between client and our controller
- We use an **observer pattern** to handle state changes in the Client * UI. This way whenever we change states for ex leaves a channel, We notify our observers (UI) that we need to update our channel list. We also have a **visitor pattern** setup in the UI in order to pattern match on the state changes and handle those accordingly.
- We use a **facade pattern** in the client module, Client is a facade for the client*side operations, hiding the complexities of the ClientCommunicationManager & ClientChannelRecord, i.e provides a simplified interface for the client*side operations.

UML Below:

https://drive.google.com/file/d/1JFjcGW4i9rPGsPwExs3kspswlE4KX0_4/view?usp=sharing if pictures isn't clear enough


Facade
Client
+ field: type
+ method(type): type
<<ClientSubject>>
«interface»
Runnable
Threads
java.io.ObjectOutputBuffer
java.io.ObjectInputBuffer
java.net.Socket
java.io.ServerSockets
Extends
Server
- ServerSocket serverSocket
- Server server
- Map<String, ChatChannel> channel
+ createServerInstance (int port)
-Server(int port)
-StartListning()
-getOrCreateChannel(String channelName)
ClientHandler
clientSocket: Socket
input: ObjectInputStream
output: ObjectOutputStream
server: Server
currentChannel: ChatChannel
ClientHandler(Socket, Server): ClientHandler
run(): void
joinChannel(String): void
sendMessage(Message): void
ClientCommunicationManager
-socket:Socket
-in:ObjectInputStream
-out:ObjectOutputStream
-host:String
-port:int
-visitor:ClientVisitor
+ClientCommunicationManager
String host,
int port,
ClientChannelRecord record,
List<ClientObserver> observers
+run:void
-handleMessage(ClientMessage m):void
+createChannel(user,channel,password):void
+sendMessage(user,channel,message):void
+joinChannel(user,channel,password):void
+leaveChannel(user,channel):void
+disconnect(user,channel):void
-sendMessageToServer(servermessage):void
ClientChannel
-name:String
-usersInChannel:List<String>
-history:StringBuilder
+ClientChannel
(
String channel Name
)
+ClientChannel
(
ClientChannel clientChannel
)
+ClientChannel
(
String ,name,
String[] usersInChannel
StringBuilder history
)
+recordMessage(message):void
+setUsers(usersInChannel):void
+getUsers():List<String>
+addUserToChannel(user):void
+removeUser(user):void
+getName():String
+setHistory(history):void
+getHistory():StringBuilder
+setName(name):void
ClientChannelRecord
-channels:List< ClientChannel>
-currentChannel:ClientChannel
+ClientChannelRecord()
+switchToChannel(channel):void
+switchToNextChannel():void
+recordMessageInChannel(message,channel):void
+addNewChannel(channel):void
+removeChannel(channel):void
+lastChannels():void
+getter..
ChatChannel
-String name;
Set<ClientHandler> clients
+ GetName() : void
+ ChatChannel(String name
void
+addClient(ClientHandler):
void
+removeClient(ClientHandle
+broadcastMessage,
ClientHandler>:void
Abstract<<Message>>
- timeStamp : TimeStamp
+getters()
ServerMessage
SendMessageInChannelCommand
JoinChannelCommand
LeaveChannelCommand
CreateChannelCommand
ClientMessage
MessageInChannel
CreateChannelResponse
ErrorResponse
JoinChannelResponses
LeaveChannelResponse
UIMessage
MessageInChannel
UpdateChannels
Error
ClientVisitor
-channelRecord:ClientChannelRecord
-observers:List<ClientObserver>
+ClientVisitor
(
ClientChannelRecord record
List<Observer> observers
)
+handleMessageInChannel m):void
+handleCreateChannelResponse m):void
+handleErrorResponse m):void
+handleJoinChannelResponse m):void
+handleLeaveChannelResponse m):void


<<Interface>>
ViewFactory
+createView() : : IView
StandardViewFactory
+createView() : : IView
"creates"
UIController
- view: IView
- reference: Client
- handleMessageDecorator: HandleMessageDecorator
- channelController: UIChannelController
+UIController(view: IView, ref: Client)
WindowController
- view: IView
- reference: Client
+WindowController(view: IView, reference: Client)
-handleWindowClosing() : : void
UIChannelController
- view: IView
- reference: Client
+UIChannelController(view: IView, reference: Client)
+joinChannel() : : void
+createChannel() : : void
+setNickName(name: String) : : void
UIClientObserver
- view: IView
- handleMessageDecorator: HandleMessageDecorator
+UIClientObserver(view: IView)
+update(message: UIMessage) : : void
+loadHistory(history: StringBuilder) : : void
<<Interface>>
IView
+startArea() : : void
+showChatArea() : : void
+ ButtonMethods(ActionListener) :: void
+updateChannelList(channels: List<String>, currentChannel: String) : : void
+getInputText() : : String
+appendChatText(text: String) : : void
+appendChatText(text: DisplayMessage) : : void
+appendChatText(ft: TextFormat) : : void
+getChannelNameAndPasswordInput(type: String) : : String[]
+removeChannelFromList(channelName: String) : : void
+clearInputText() : : void
+getNickNameField() : : String
+displayErrorMessage(message: String) : : void
+showNotification(message: String) : : void
+getChannelList() : : DefaultListModel<String>
+addWindowExitListener(listener: WindowListener) : : void
+closeWindow() : : void
StandardView
+StandardView()
+startArea() : : void
+showChatArea() : : void
+updateChannelList(channels: List<String>, currentChannel: String) : : void
+getChannelList() : : DefaultListModel<String>
+showHistory(history: StringBuilder) : : void


+removeChannelFromList(channelName: String) : : void
+clearInputText() : : void
+getNickNameFeild() : : String
+displayErrorMessage(message: String) : : void
+showNotification(message: String) : : void
+getChannelList() : : DefaultListModel<String>
+addWindowExitListener(listener: WindowListener) : : void
+closeWindow() : : void
StandardView
+StandardView()
+startArea() : : void
+showChatArea() : : void
+updateChannelList(channels: List<String>, currentChannel: String) : : void
+getChannelList() : : DefaultListModel<String>
+showHistory(history: StringBuilder) : : void
+changeChannel(channelName: String) : : void
+displayErrorMessage(message: String) : : void
+showNotification(message: String) : : void
+getChannelNameAndPasswordInput(type: String) : : String[]
+addCreateButtonListener(listener: ActionListener) : : void
+getNickNameFeild() : : String
+ ButtonMethods(ActionListener listener) :: void
+addChannelListSelectionListener(listener: ListSelectionListener) : : void
+addLeaveChannelButtonListener(listener: ActionListener) : : void
+getInputText() : : String
+appendChatText(text: String) : : void
+appendChatText(ft: TextFormat) : : void
+appendChatText(text: DisplayMessage) : : void
+removeChannelFromList(channelName: String) : : void
+clearInputText() : : void
+addWindowExitListener(listener: WindowListener) : : void
+closeWindow() : : void
ViewDecorator
- decoratedView: IView
+ViewDecorator(decoratedView: IView)
+startArea() : : void
+showChatArea() : : void
+ ButtonMethods(ActionListener listener) :: void
+getInputText() : : String
+appendChatText(text: String) : : void
+appendChatText(text: DisplayMessage) : : void
+appendChatText(ft: TextFormat) : : void
+clearInputText() : : void
+getNickNameFeild() : : String
+getChannelNameAndPasswordInput(type: String) : : String[]
+displayErrorMessage(message: String) : : void
+showNotification(message: String) : : void
+updateChannelList(channels: List<String>, currentChannel: String) : : void
+changeChannel(channelName: String) : : void
+addChannelListSelectionListener(listener: ListSelectionListener) : : void
+removeChannelFromList(channelName: String) : : void
+getChannelList() : : DefaultListModel<String>
+showHistory(history: StringBuilder) : : void
+addWindowExitListener(listener: WindowListener) : : void
+closeWindow() : : void
DecryptionDecorator
+DecryptionDecorator(decoratedView: IView)
+appendChatText(text: DisplayMessage) : : void
-decrypt(text: String) : : String
HandleMessageDecorator
+HandleMessageDecorator(decoratedView: IView)
+handle(e: DisplayError) : : void
+handle(m: DisplayMessage) : : void
+handle(u: UpdateChannels) : : void
TimestampDecorator
+TimestampDecorator(decoratedView: IView)
+appendChatText(text: DisplayMessage) : : void
<<interface>>
UIMessageVisitor
+handle(e: DisplayError) : : void
+handle(m: DisplayMessage) : : void
+handle(u: UpdateChannels) : : void