Team _Undefined

Secure Chat Service

Members: Samuel Launt, Tyler Paulsen, Lai Chung Lau

Table of Contents

Illustration Index

Index of Tables

What's new?	Date changed
Doc created, format established	3/8/16
Added more: UMLs, client/server info, GUI guidelines	3/10/16
Added features to protocol, updated protocol diagrams, added to GUI spec	3/29/16
Updated protocol and fixed some inconsistency error. Also added Port information	4/5/16
Updated protocol and adjusted diagrams accordingly	4/6/16
Added GUI UML	4/12/16
Modified client server UML and updated protocol	4/13/16

Table 1: Revision History

Overview

The Goal of this project is to create a secure chat service that will protect the security of its users from the prying eyes of the internet. The main requirements are having a hybrid infrastructure, peer to peer encrypted communication, custom protocol, functional GUI, the ability to have multiple chat sessions open at once, and have an encrypted databases in server/client.

Planned Implementation

The project will be implemented in Java. We chose Java because we are all experienced with it and it has a large built-in library of features. Also it cross platform, so it can be run anywhere.

We are going to use, what we call a hybrid system. It is going to use a client and a server, but the server is only going to handle overhead. Things like keeping track of all the client IPs and friend requests. The server is basically a DNS server with a little more functionality. We chose this architecture because this seemed more secure than a totally centralized system and less complicated than an entirely distributed system. For security the client is going to authenticate with the server and report to the server its IP.

To start a conversation, the user needs to already be friends with the other user. The first thing the client is going to do is request of the IP of the other user from the server. Then the client is going to send a start conversation message to the server to give to the client. Then relatively simultaneously each client will start a SSL encrypted TCP session with each other. This is a method called TCP Hole Punching and is used to “break” through NATs that each user is most likely behind.

There is going to be a need of databases for this project. We have chosen H2 has our database. H2 is a pure Java database and can be fully encrypted. Also it is very easy to work with.

Final Goals

- Hybrid model infrastructure
- Peer to peer encrypted communication
- GUI
- Multiple conversations at once
- Encrypted client/server databases
- Capable of dealing with NAT/Firewalls

Stretch Goals

- Group chat rooms
- Modern, minimalist interface
- Password Requirements
- Automated install script
- File transfer

Protocol

The protocol is used when communications over the network are necessary. For the protocol we are going to use a minimized textual encoding, which means we are going to print the strings/characters to the socket instead of printing them in binary. The reason for this is that it is easier to work with and we aren't overly concerned with bandwidth usage.

Hash

Mid Development we Noticed a flaw in the design of the program. Anyone could use the defaults for SSL socket in java and connect to the server. So as a unique ID for each user we are using SHA-256 for the hashing algorithm. The input for this algorithm is the password to the users database that is created. This way even if someone connects with the defaults the messages should just be ignored.

Port

The ports used in the program are defined as follows: Port 5432 is used for connecting to the server. For the peer to peer communication we are using non used dynamic ports. Specificly when creating the socket we are using port 0 which isn't a real port and java will select a port from the dynamic rang to create the socket with.

Protocol Definition

Client to Client

- Message: m <From> <To> <Message>
- Start conversation: s
    - **NOTE –** When s is without parameters this isn't a real message sent, it only represents the opening and connecting of the sockets
- End conversation: q

Client to Server

- Friend request: f <From> <Hash> <To> <Status>
    - Status' 0-pending, 1-accept, 2-reject
- Requesting username: r <Username> <Hash>
    - Used for establishing an username on the service
- Starting connection to server: s
    - **NOTE –** When s is without parameters this isn't a real message sent, it only represents the opening and connecting of the sockets
- Rejoin Service: J <Username> <Hash>
- Ending connection to server: Q <Username> <Hash>
- Start conversation with client: S <From> <From_Hash> <To>
- Getting IP of user: G <From> <From_Hash> <To>
- conversation reject: Z <From> <From_Hash> <To>
- Remove friend: M <from> <from_Hash> <friend>

Server to Client

- Friend request: f <From> <To> <Status>
    - Status' 0-pending, 1-accept, 2-reject
- Return IP of a user: g <Username> <IP>
- General error message: e <error message>
- Conversation reject: Z <From> <To>
- Log on Success: R
- Log on Response: C <User> <status>
- Remove Friend: M <From> <Friend>

Protocol Flows

Flow diagrams of how the protocol works.

Joining Service:

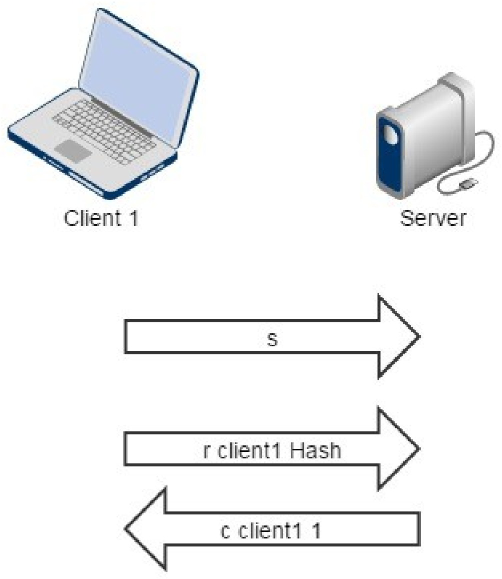

Illustration 1: Joining Service

Starting up client:

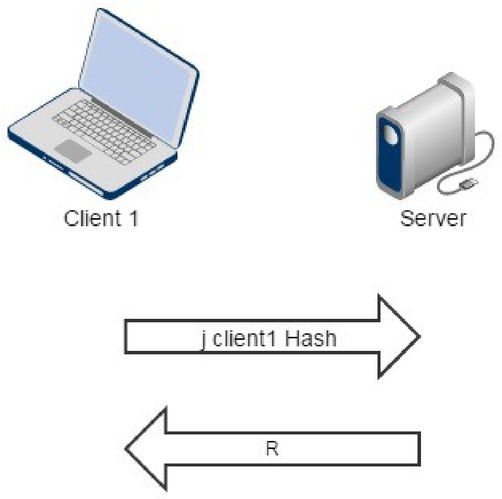

Illustration 2: Starting Program

Ending connection with server:

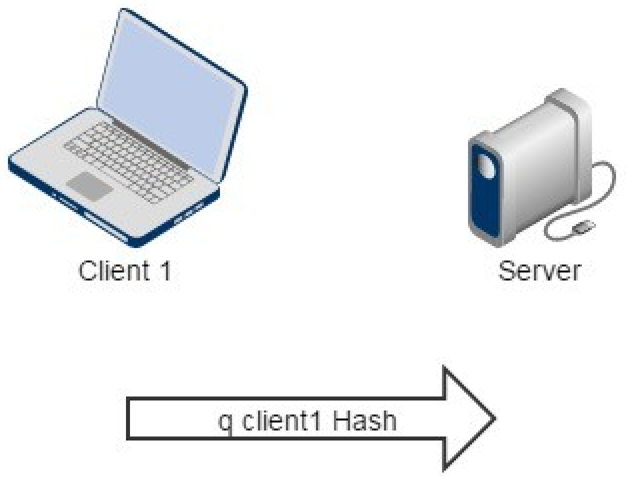

Illustration 3: Ending Program

Friend requests:

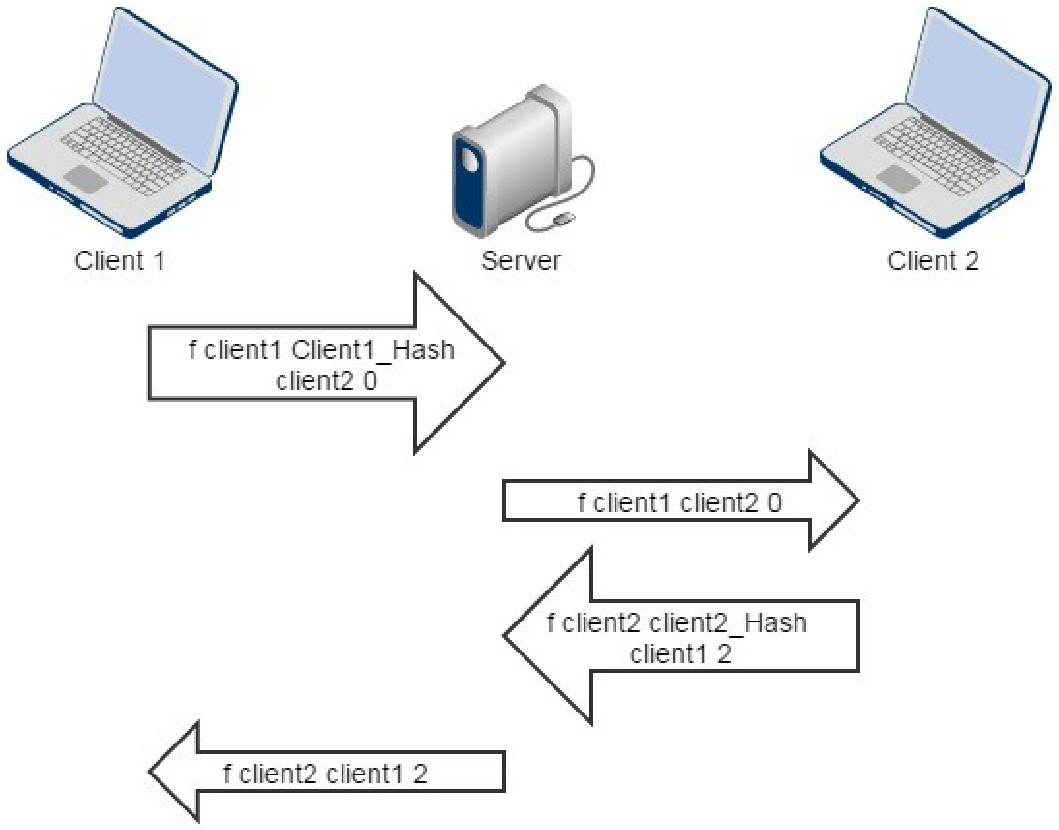
Client 1
Server
Client 2
f client1 Client1_Hash client2 0
f client1 client2 0
f client2 client2_Hash client1 2
f client2 client1 2

Illustration 4: Friend Request Reject

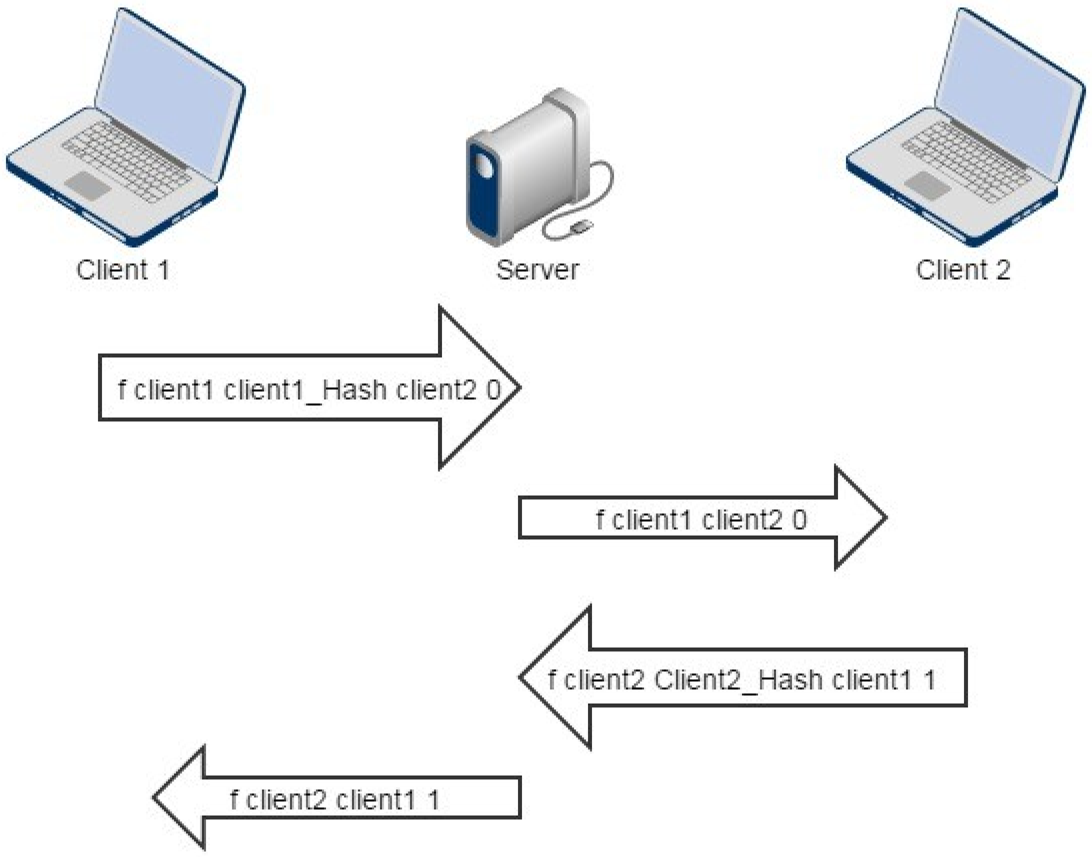
Client 1
Server
Client 2
f client1 client1_Hash client2 0
f client1 client2 0
f client2 Client2_Hash client1 1
f client2 client1 1

Illustration 5: Friend Request Accept

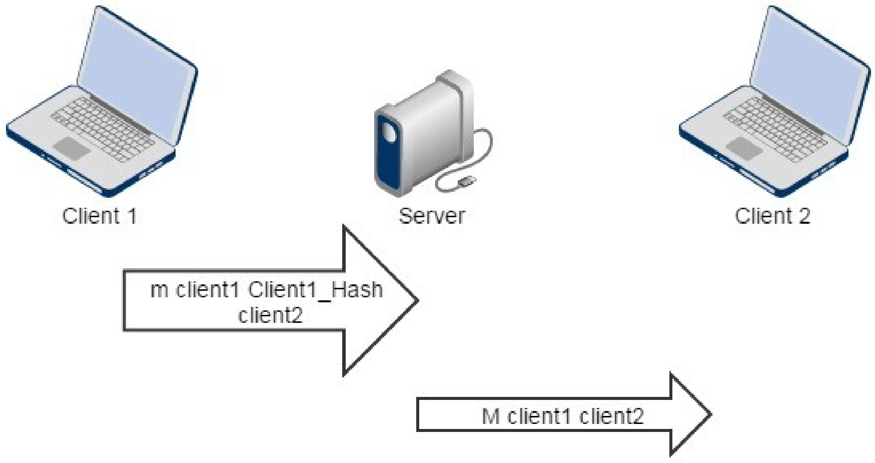

Illustration 6: Remove Friend

Starting Conversation:

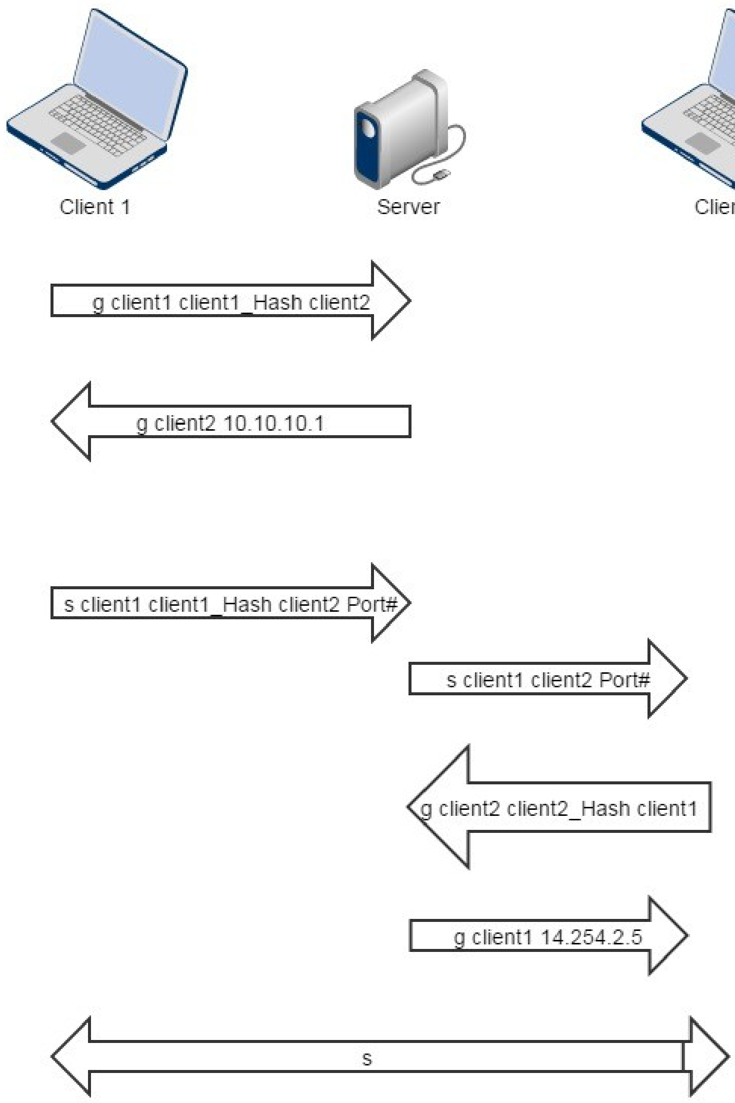
Client 1
Server
Client 2
g client1 client1_Hash client2
g client2 10.10.10.1
s client1 client1_Hash client2 Port#
s client1 client2 Port#
g client2 client2_Hash client1
g client1 14.254.2.5
s

Illustration 7: Starting conversion

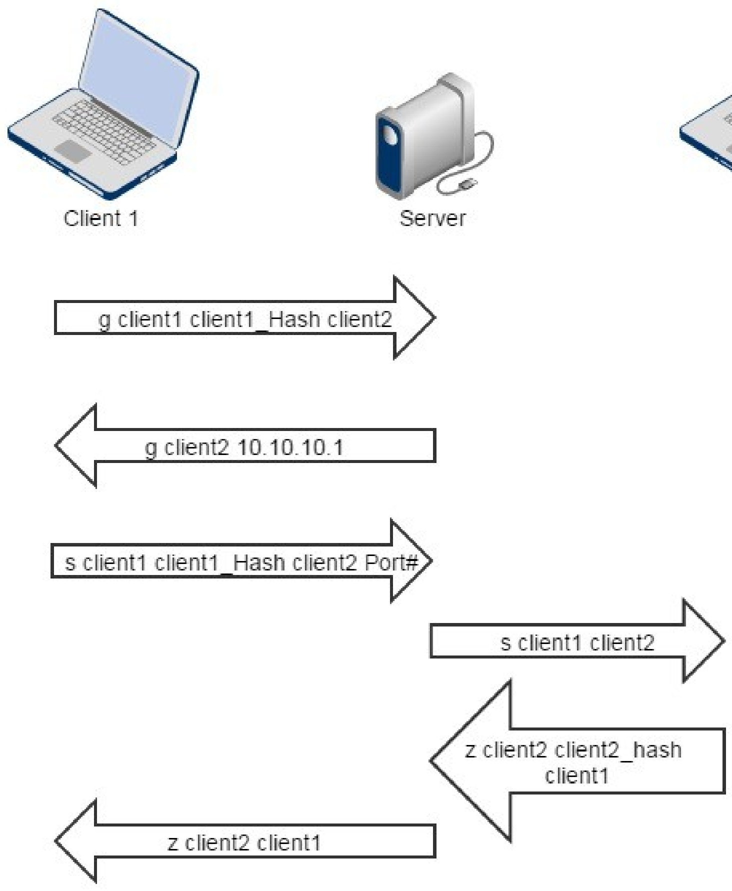
Client 1
Server
Client 2
g client1 client1_Hash client2
g client2 10.10.10.1
s client1 client1_Hash client2 Port#
s client1 client2
z client2 client2_hash client1
z client2 client1

Illustration 8: Conversation Decline

Connecting to Peer:

From the diagrams it should be clear on how the protocol should work, but it is note worthy that a peer only try to establish a connection when the client get the s message. The first person that receives the s message has the option to decline the conversation and no sockets are opened.

Ending Connections:

In all cases when ending the TCP connections either with the client or with the server, the socket is only closed when you receive the message. This means the first person to receive the message must send another message back, this both acknowledges that the first person got it and it allows both sides to shutdown gracefully.

Server Design

The server is designed to be a hybrid system where all communications (messaging) are only between peers. There are cases where some messages go through the server, these would be things like Friend requests. This was a design decision, because you can only request a persons IP through the server if your friends with them, so we wanted the friend requests to go through the server as a way to protect the IPs of users and to only allow people that they are friends with to have access to there current IP.

Server UML

This is the full server UML. From this it should be easy to see how the different classes interact with each other.

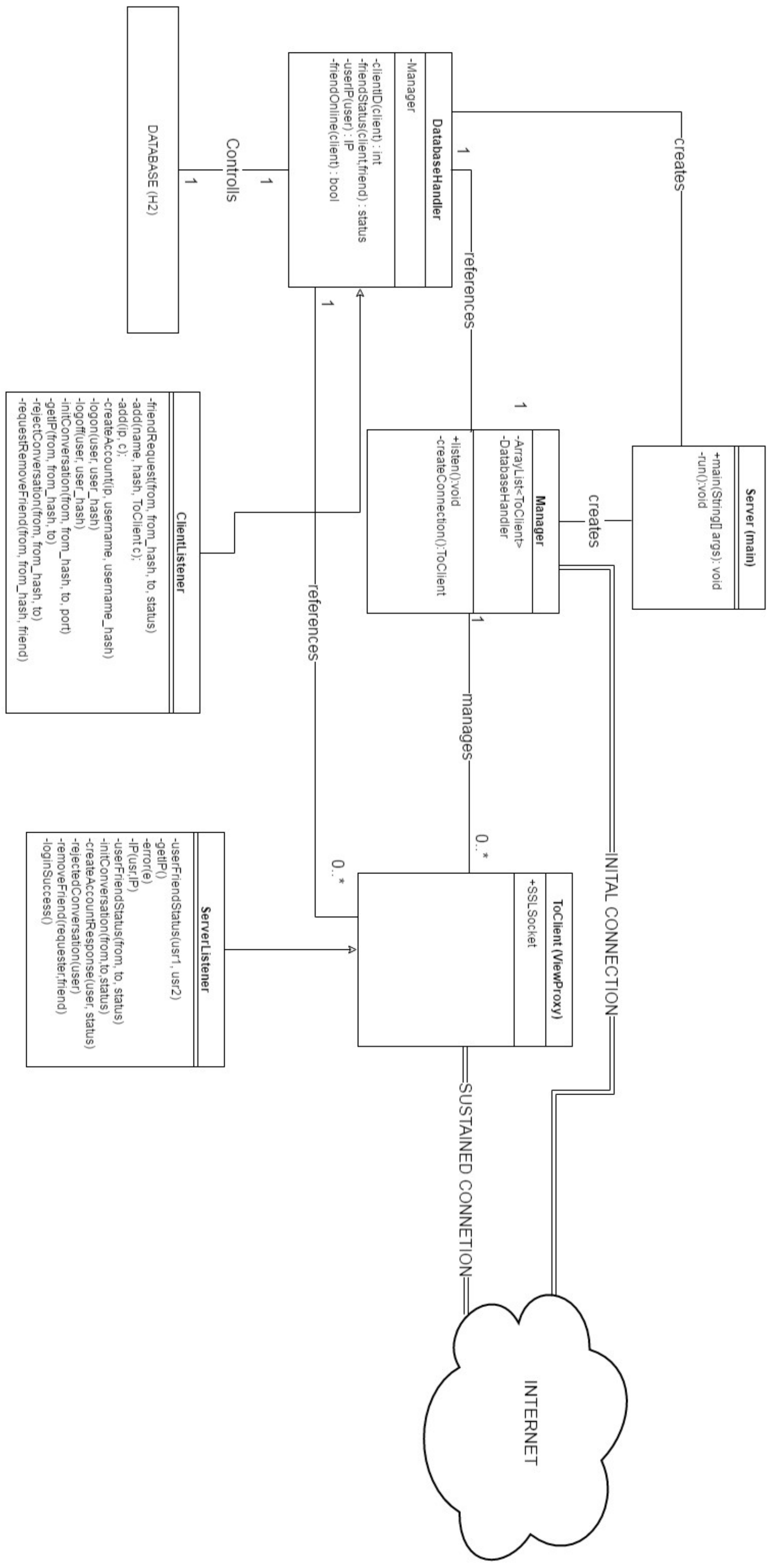
DatabaseHandler
-Manager
-clientID(client) : int
-friendStatus(client,friend) : status
-userIP(user) : IP
-friendOnline(client) : bool
DATABASE (H2)
Controlls
1
1
creates
1
references
1
Manager
-ArrayList<ToClient>
-DatabaseHandler
+listen():void
-createConnection():ToClient
creates
Server (main)
+main(String[] args): void
-run():void
ClientListener
-friendRequest(from, from_hash, to, status)
-add(name, hash, ToClient c);
-add(ip, c);
-createAccount(ip, username, username_hash)
-logon(user, user_hash)
-logoff(user, user_hash)
-initConversation(from, from_hash, to, port)
-getIP(from, from_hash, to)
-rejectConversation(from, from_hash, to)
-requestRemoveFriend(from, from_hash, friend)
1
references
1
manages
0..*
0..*
INITAL CONNECTION
ToClient (ViewProxy)
+SSLSocket
ServerListener
-userFriendStatus(usr1, usr2)
-getIP()
-error(e)
-IP(usr,IP)
-userFriendStatus(from, to, status)
-initConversation(from,to,status)
-createAccountResponse(user, status)
-rejectedConversation(user)
-removeFriend(requester,friend)
-loginSuccess()
SUSTAINED CONNETION
INTERNET

Illustration 9: Server UML

Important Relations

It should be noted that all the main class does is creates instances of Manager, and DataBaseHandler. The Manager class is holding the initial SSL socket listening for the initial connections. When a connection comes in it then creates a ToClient and gives it the SSL socket.

Interfaces

The 2 interfaces in the program (ClientListener and ServerListener) are abstractions of the protocol used to communicate. So along as a class implements these interfaces that program should be able to communicate using our protocol.

DatabaseHandler

The DatabaseHandler is the class that implements ClientListener. Basically this makes the DatabaseHandler one of the major parts of the server. Taking in client messages, processing them, and responding accordingly. Lastly the DatabaseHandler holds the connection to the H2 database and makes the corresponding SQL queries to it.

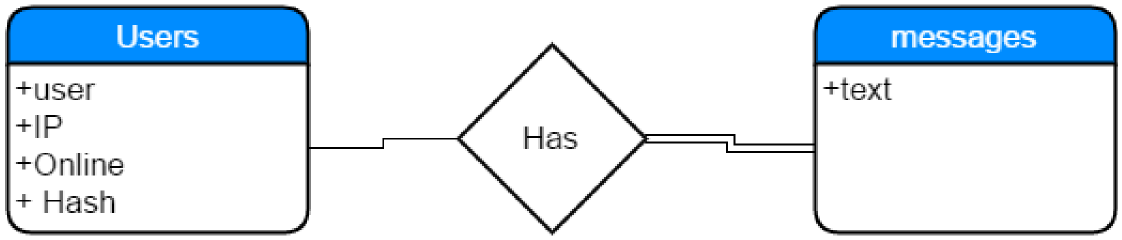

Illustration 10: Server Database

From Illustration 9, you should see some things. The first is that all that is stored is the usernames associated with this server, the last known IP of the user, and the current online status of the user. This database has one relation, and this is to another table messages, where a user (PK) has messages. This is only for when the user is offline and the user gets a friend request. Then the next time that user goes online the server will forward all messages that have been in limbo. This will also let us expand functionality later, if needed.

Client Design

The client is designed to have a sustained TCP connection to the server and peer to peer TCP connections to what ever client they are talking to at the moment.

Client UML

This is the full client UML. From this it should be easy to see how the different classes interact with each other.

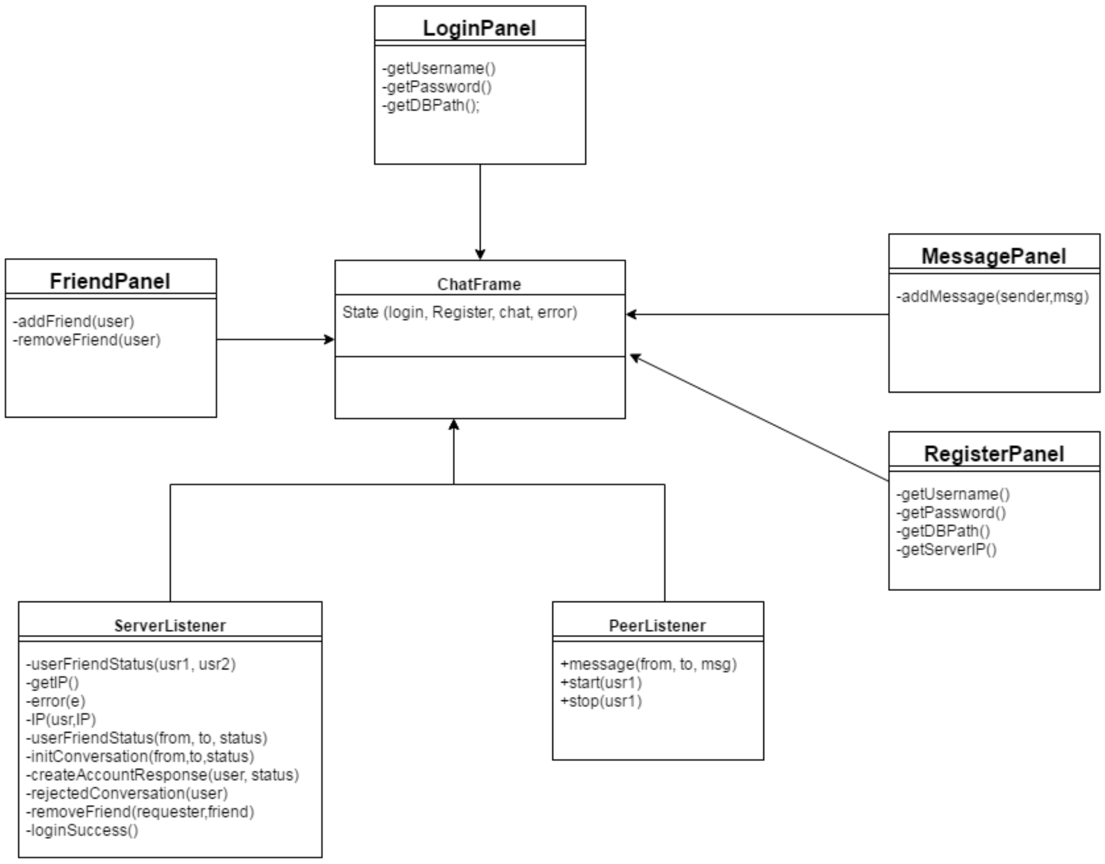

Illustration 11: GUI UML

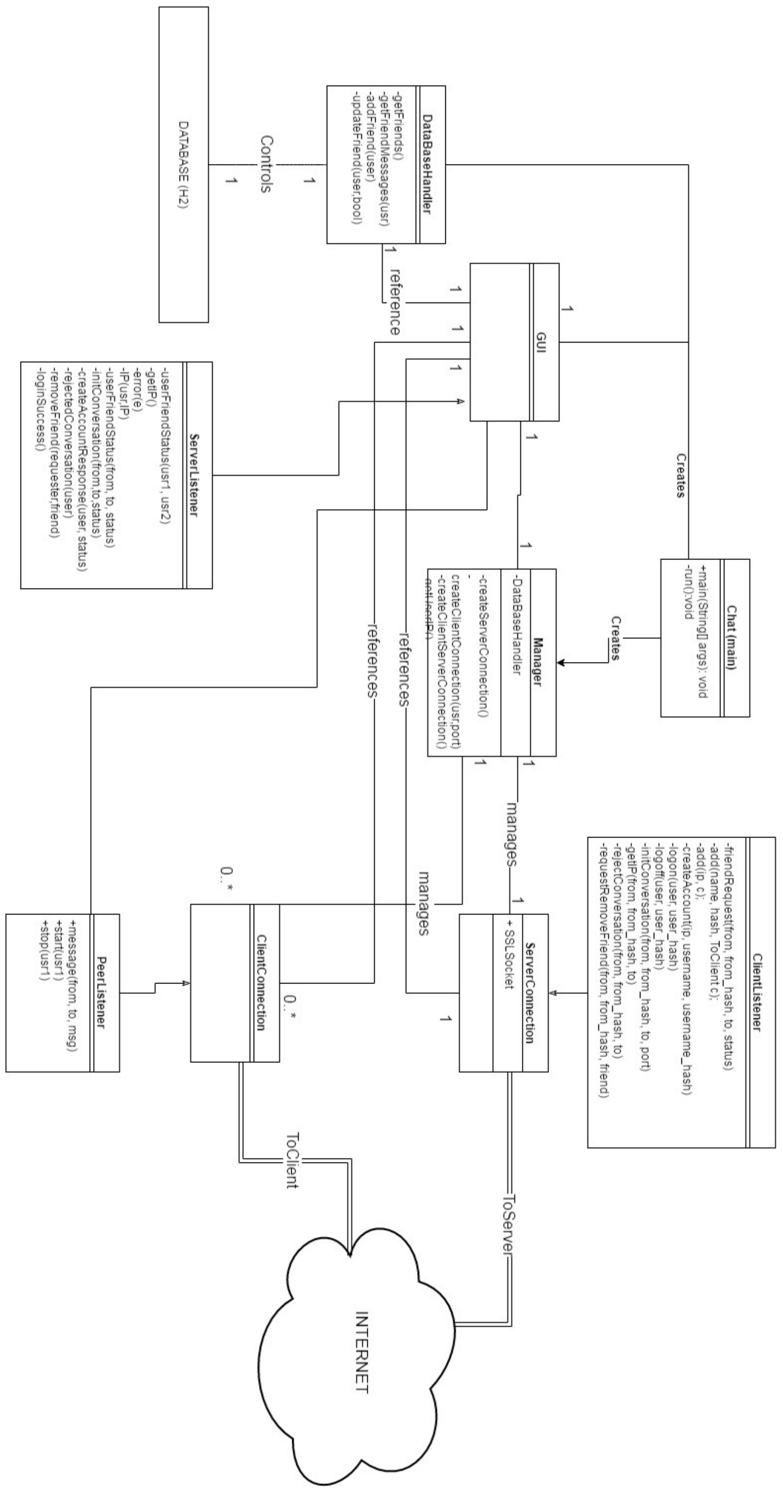
DATABASE (H2)
Controls
DataBaseHandler
-getFriends()
-getFriendMessages(usr)
-addFriend(user)
-updateFriend(user,bool)
reference
GUI
ServerListener
-userFriendStatus(usr1, usr2)
-getIP()
-error(e)
-IP(usr,IP)
-userFriendStatus(from, to, status)
-initConversation(from,to,status)
-createAccountResponse(user, status)
-rejectedConversation(user)
-removeFriend(requester,friend)
-loginSuccess()
Chat (main)
+main(String[] args): void
-run():void
Creates
Manager
-DataBaseHandler
-createServerConnection()
-createClientConnection(usr,port)
-createClientServerConnection()
references
manages
ClientListener
-friendRequest(from, from_hash, to, status)
-add(name, hash, ToClient c);
-add(ip, c);
-createAccount(ip, username, username_hash)
-logon(user, user_hash)
-logoff(user, user_hash)
-initConversation(from, from_hash, to, port)
-getIP(from, from_hash, to)
-rejectConversation(from, from_hash, to)
-requestRemoveFriend(from, from_hash, friend)
ServerConnection
+ SSLSocket
ClientConnection
PeerListener
+message(from, to, msg)
+start(usr1)
+stop(usr1)
0..*
1
ToClient
ToServer
INTERNET

Illustration 12: Client UML

Important relations

Note: Illustration 10 just replaces the GUI class from Illustration 11.

Like in the server the main class only creates instances of a few classes. In the case of the client it is the GUI, DatabaseHandler, and the Manger. The Managers job is to create the connection to the server and create connections to the client when needed.

DatabaseHandler

The client database is simpler when compared to the server database but is just as important. In this case the client database is the clients ID and without it you are no longer that person.

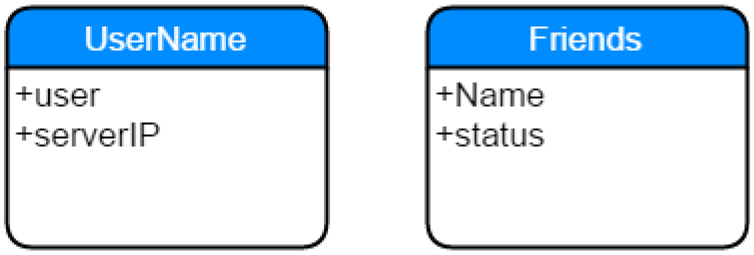

Illustration 13: Client Database

There are 2 tables from the database. One only holds the user's name, this is for data persistence. The other is the larger table, it holds your friends (there username) and the status is either or not the friend is pending.

Interfaces

The client implements the same interfaces as the server except for one difference. That would be there is an extra interface called PeerListener. This is the interface for communicating with other peers. It is implemented in 2 different locations, one in the client connection and in the GUI.

Message

From the UML there is a message class coming off of the GUI class. This class represents a whole conversation and with hold all of the messages for a conversation. This is will make displaying the messages in the GUI as well as keeping track of the history of the conversation.

GUI

The GUI is one of most complicated parts of the whole system. It has many roles, from calling methods in the database, to acting as the server in the peer to peer communication. It is also the first thing that the user actually sees. The first iteration is to be functional and not pretty, but this might change through the evolution of the program. Here are the basic GUI views: Login Screen, Home Screen, Error Screen, and new account creation screen.

GUI Outline

Login:

This is the View when the user first starts up the program.

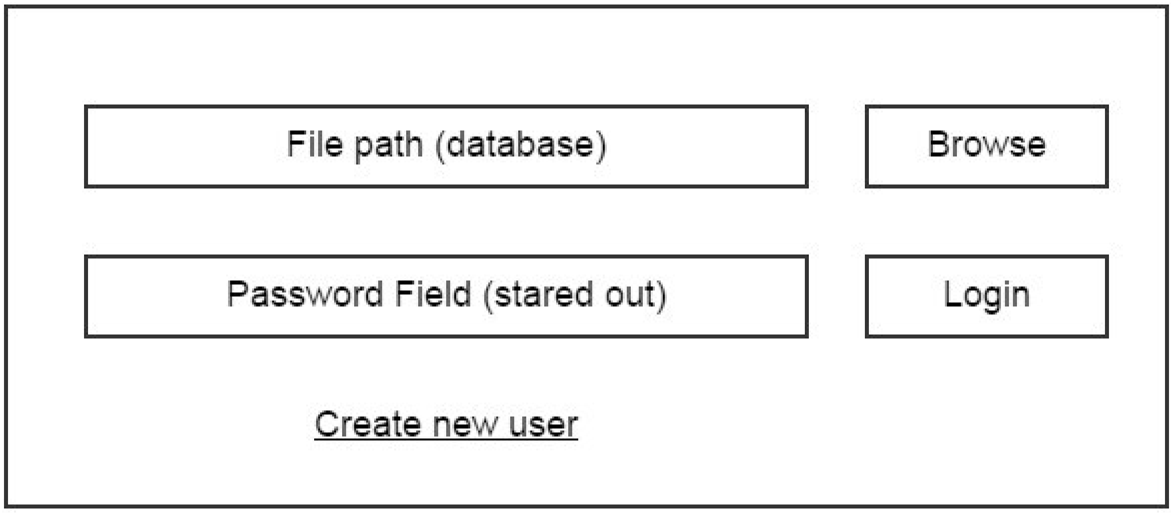

Illustration 14: Login Screen

Error:

This is the screen a user will see when some things goes wrong. For example when a username has already been taken or when you try to login with specifing a path to a database file.

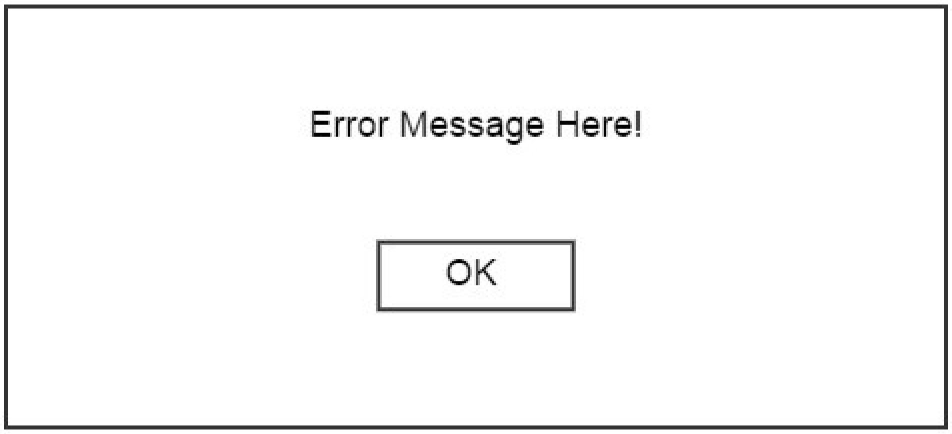

Illustration 15: Error Screen

Account Creation:

This is the form that a user needs to fill out to create a new user on the service.

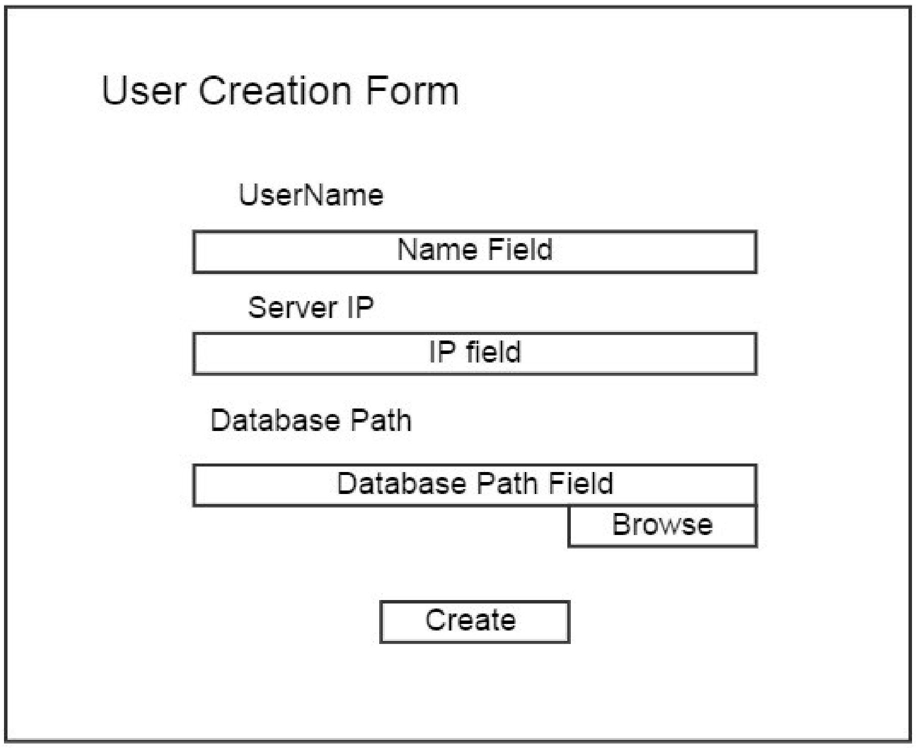

Illustration 16: User creation Form

Home Screen:

These are some of the basic states of a logged in home screen.

Starting Home Screen:

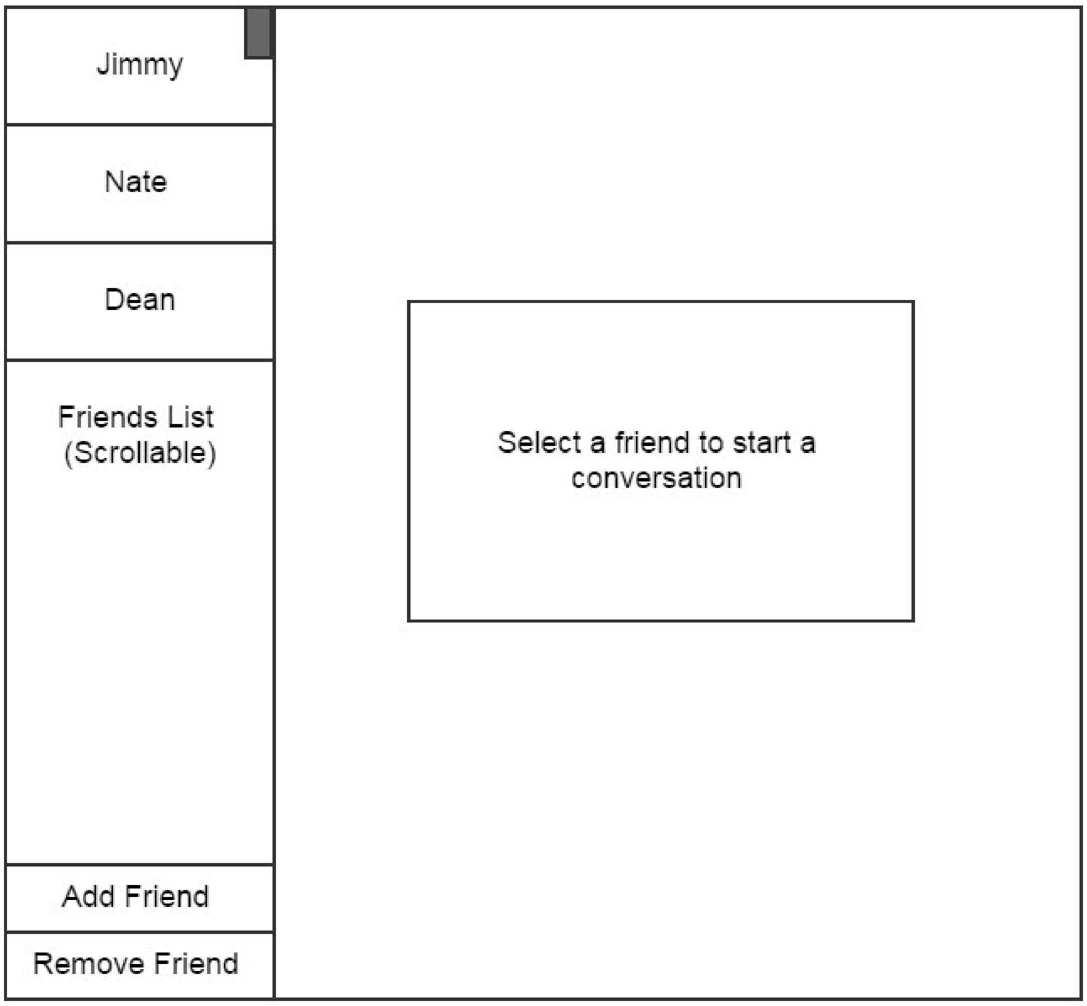
Jimmy
Nate
Dean
Friends List (Scrollable)
Add Friend
Remove Friend
Select a friend to start a conversation

Illustration 17: Start Home Screen

After the First Screen and the user clicks on a friend:

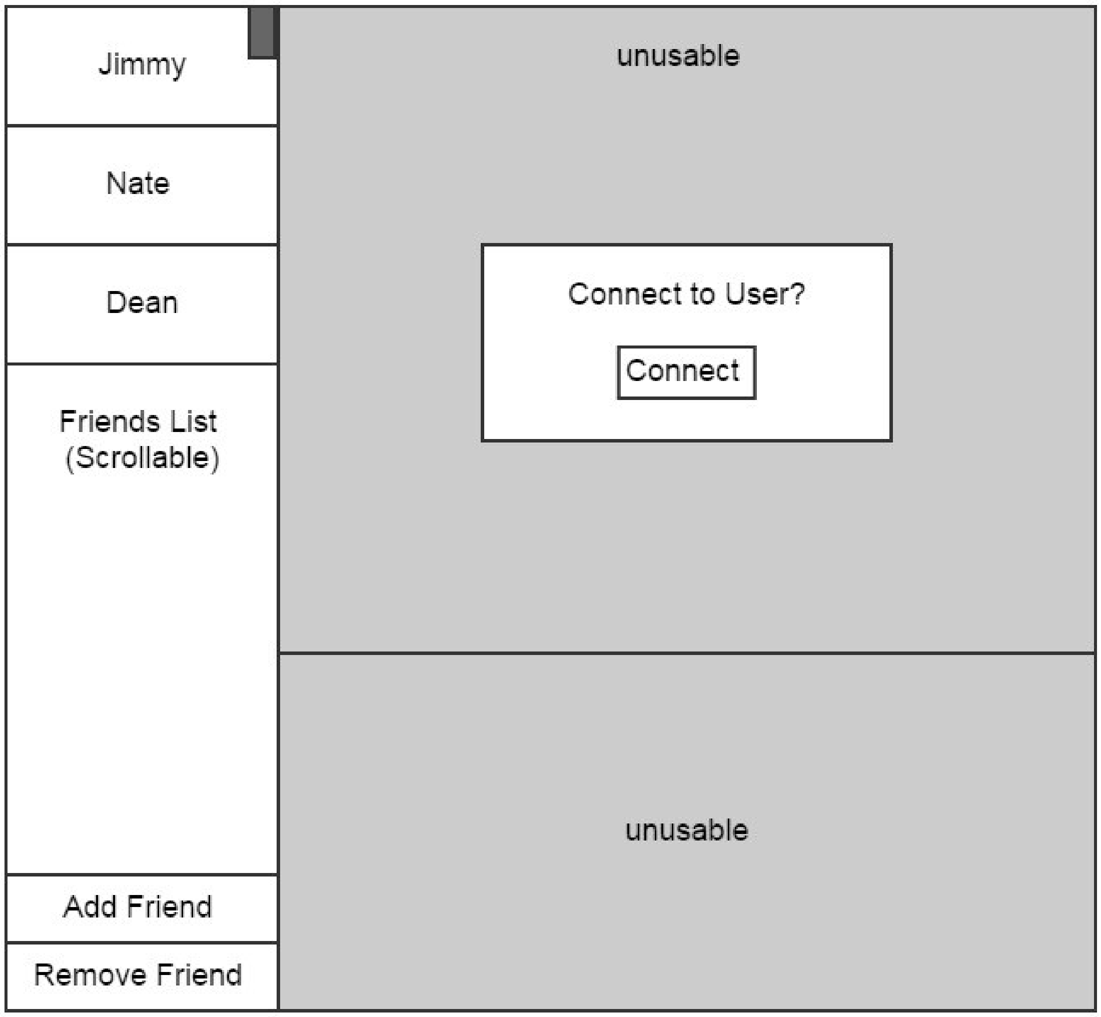

Illustration 18: Home Screen: First Click

The Home screen during a conversation:

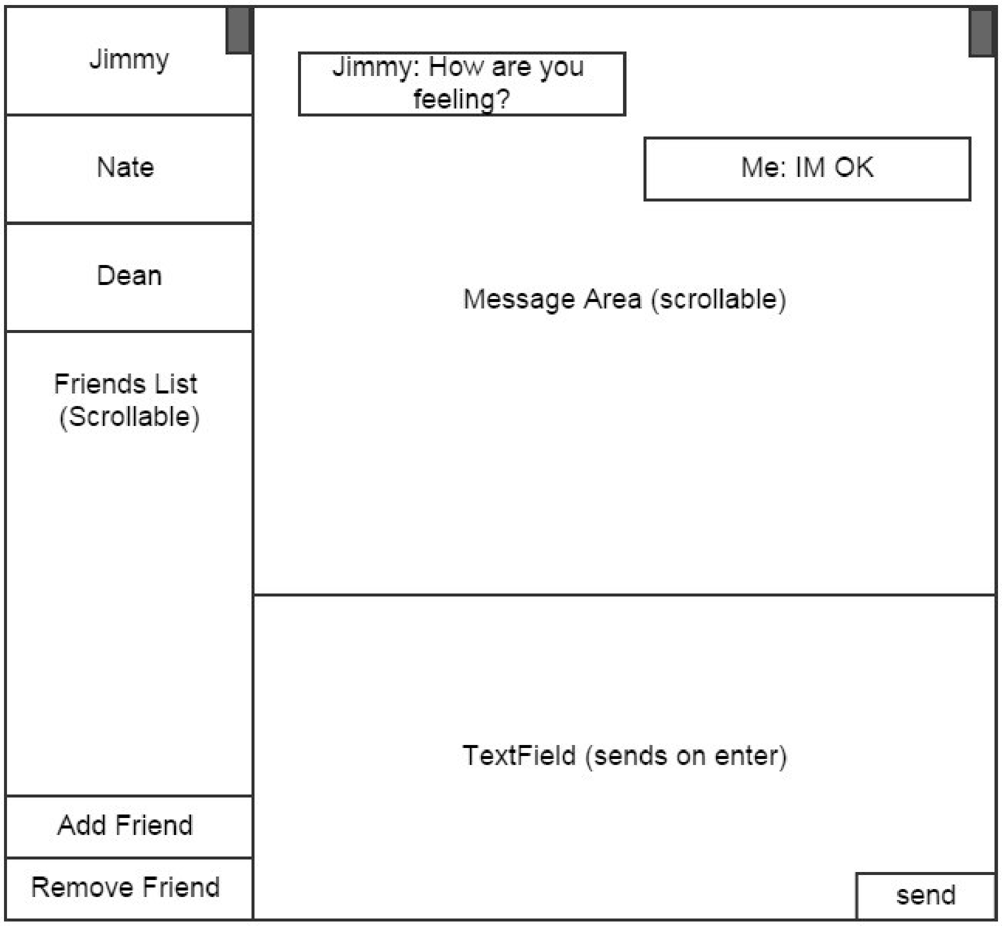
Jimmy
Nate
Dean
Friends List (Scrollable)
Add Friend
Remove Friend
Jimmy: How are you feeling?
Me: IM OK
Message Area (scrollable)
TextField (sends on enter)
send

Illustration 19: Home Screen Conversation

Lastly a notification on the Home Screen:

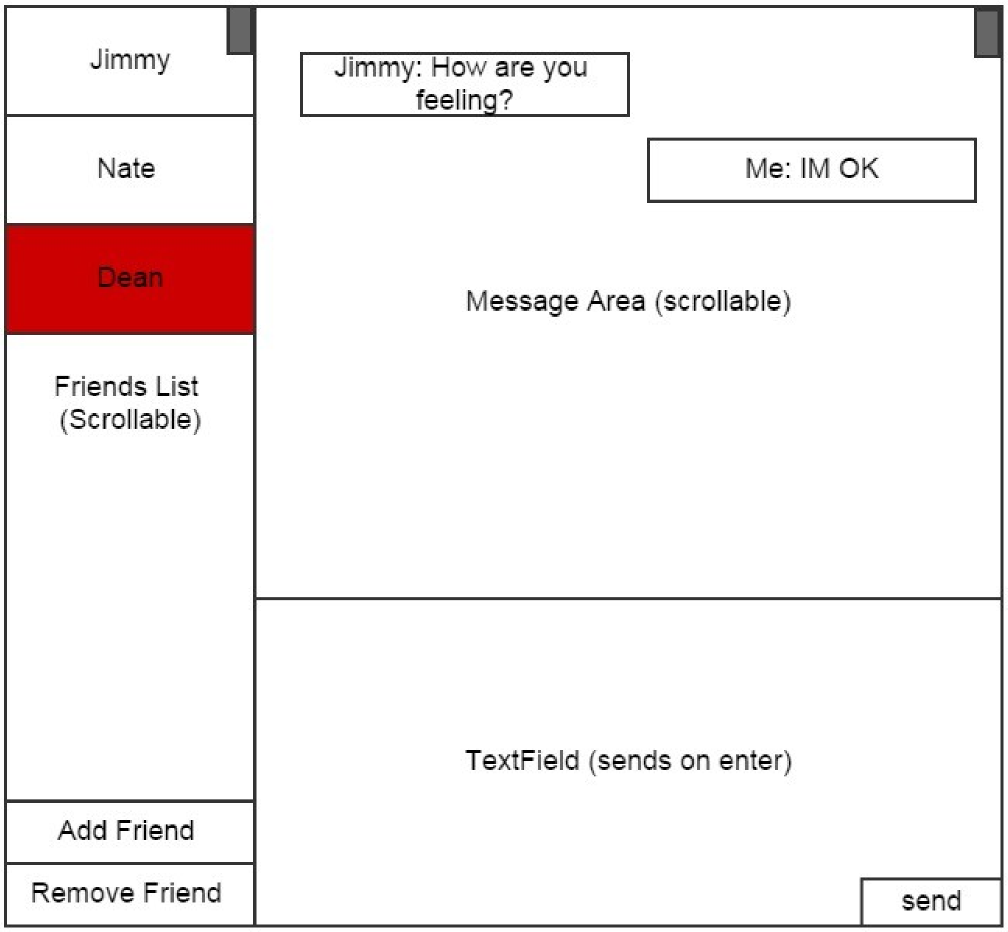
Jimmy
Nate
Dean
Friends List (Scrollable)
Add Friend
Remove Friend
Jimmy: How are you feeling?
Me: IM OK
Message Area (scrollable)
TextField (sends on enter)
send

Illustration 20: Home Screen Notification