ChatBot
A chat tool

Version 1.0

PROGRAMMERS:

Martino Caldarella,

Hanqi Chen,

Ragui Halim,

Hoon Lee,

Jacobis Soriano,

Daniel Truong

BitBots Software Developers from UCI

Table of contents:

Glossary 2
1. Server Software Architecture Overview 4
1.1. Main data types and structures 4
1.2. Major software components 4
1.3. Module interfaces 5
1.4. Overall program control flow 7
2. Client Software Architecture Overview 9
2.1. Main data types and structures 9
2.2. Major software components 9
2.3. Module interfaces 10
2.4. Overall program control flow 15
3. Installation 15
3.1. System requirements, compatibility 15
3.2. Setup and configuration 15
3.3. Building, compilation, installation 15
4. Documentation of packages, modules, interfaces 15
4.1. Detailed description of data structures 15
4.2. Detailed description of functions and parameters 17
4.3. Detailed description of the communication protocol 21
4.3.1. Explanation of communication protocol 21
4.3.2. Explanation of communication related function calls 22
5. Development plan and timeline 23
5.1. Timeline 23
5.2. Partitioning of tasks 23
5.3. Team member responsibilities 23
6. Back matter 24
6.1. Copyright 24
6.2. References 24
6.3. Index 24

Glossary

- **Address** - Identifier for a node or host on a telecommunication network. An address has a unique identifier. A network address is also known as the numerical network part of an IP address.
- **Application** - program that performs specific tasks for the user.
- **Client** - a computer or a program that, as part of its operation, relies on sending a request to another or a computer hardware or software that accesses a service made available by a server. A single client can use multiple servers.
- **Communications** - The transmission of data from one computer to another, or from one device to another. A communications device is any machine that assists data transmission. For example, modems, cables, and ports are all communications devices.
- **Contact List** - a list of contacts available to the ChatBot subscriber.
- **Data** - Data is defined as facts or figures, or information that's stored in or used by a computer.
- **GTK** -GTK+, or the GIMP Toolkit, is a multi-platform toolkit for creating graphical user interfaces.
- **GUI** - Graphical User Interface. It makes it easy to communicate with the program using graphical icons and visual indicators.
- **Host** - also called “network host”. A single device connected to a network that has an IP address and that can be the source or destination of a data packet. A host can be a client or a server.
- **Idle** - one of the three states of an individual. It indicates the individual is online but occupied . ChatBot will be delivered messages immediately but individual will not be notified.
- **Internet** - a global computer network consisting of interconnected networks using standardized communication protocols. It provides a variety of information and communication facilities,
- **IP Address** - Internet Protocol Address. A numerical label given to any device connected to a network. It used to identify a host or a network interface. It is defined as a 32 bit number.
- **Instant Message (IM)** - type of service allowing the user to exchange messages with someone in real-time over the internet.
- **Library** - a collection of premade functions that can be used in a program
- **Network** - a series of nodes interconnected by communication paths, used to transmit or receive data.
- **Node** - a device or data point on a network. A node is anything (computer, cell phone, printer...) that has an IP address.
- **Notification** - a message alerting the user that some action has taken place. Could be a new message, a friend request, a friend’s status, etc.
- **Offline** - one of three states of an individual. It indicates the individual is offline and not open for connectivity. Individual cannot send messages when in this state. Individual will be unable to see his/her message until he/she signs on again.

- **Online** - one of the three states of an individual. It indicates the individual is online and available to chat. ChatBot will be deliver messages to this individual immediately.
- **Password** - a word or phrase that allows access to a tool. For IM tools, they usually require both your account name and corresponding password for it.
- **Port** - an endpoint of communication in an operating system. A port is associated to an IP address of a host and a protocol type. Ports are identified by a 16-bit unsigned numbers.
- **Server** - a computer program or a device that provides functionality for other programs or devices, defined as clients. Servers can provide various functionalities, often called "services", such as sharing data or resources among multiple clients, or performing computation for a client. A single server can serve multiple clients
- **Socket** - an endpoint of a link between two programs running on a network. It is commonly used for client/server interaction.
- **Status** - present state of an individual. For IM, there are three: online, idle, and offline.
- **Username** - a name chosen by the user to be displayed when communicating with others on ChatBot.
- **Friend** - the relationship between 2 connected users in the chat application.
- **Window** - the user interface of the application
- **Button** - an object that can be found in a window and it reacts when you click on it. It is similar to the remote control button, but in the user interface.
- **Widget** - can be a window or a component of the window, like a button.
- **Database** - a place where we can save our data.

1. Server Software Architecture Overview

1.1. Main Data Type and Structures

- Structures:
  - **ClientAccount -** contains all the information (name, username, user ID, email, password, security question and answer, contacts list, friend requests list, font, color) about the client.
  - **Message -** contains the information about person who sent a message, and text association with person who sent a message.
  - **Chat -** contains the members list, the start and end of the conversation, and saved exchanged messages
- Arrays:
  - **Chat List -** Array of chats of the entire server.
  - **Accounts List -** Array of all the contacts in the application.
- Enumerators:
  - **bool** - has true and false values.

1.2. Major Software Components

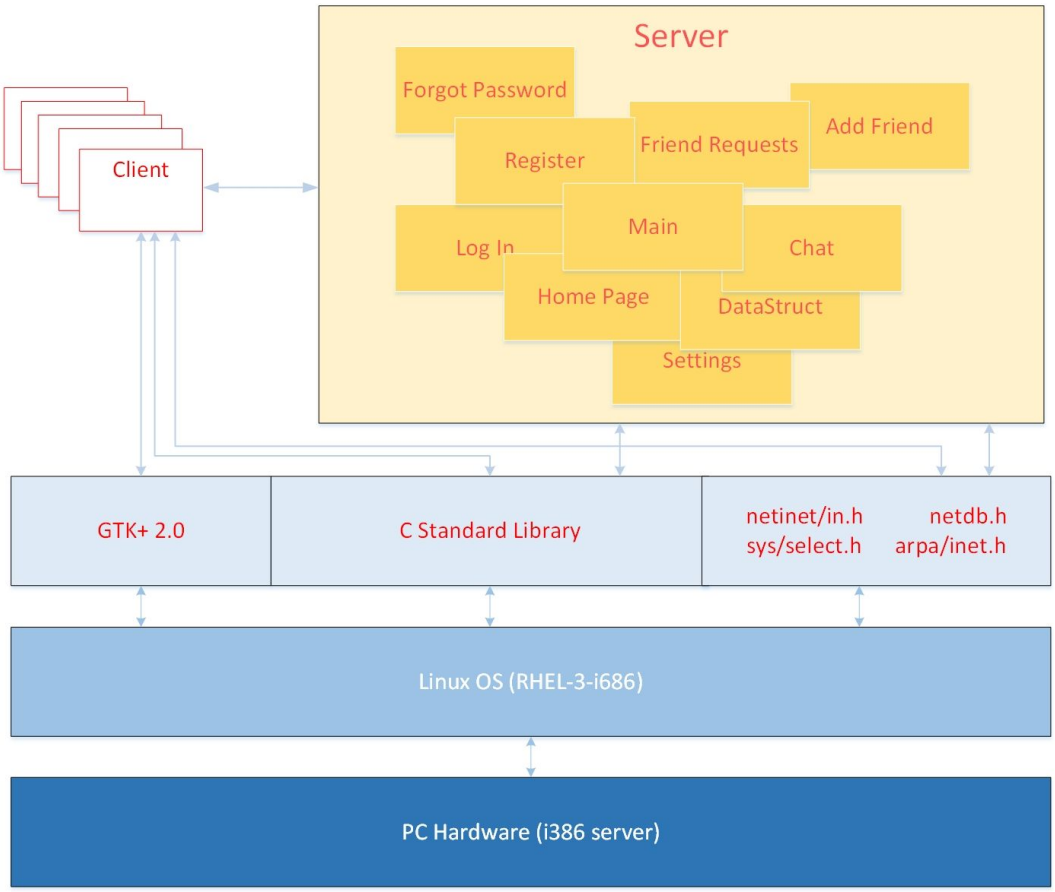

Figure 1 - server software components

The server has a main function, which calls all other functions from the modules. When a message is received by the client, a proper function is called with the corresponding parameters given in the message. Then, it writes back a message with the desired data.

1.3. Module Interfaces

- ***Login module:***
  - **`bool validLogin(const char *username, const char *password, char *errorMessage)`**
    - **Input:** username and password entered by user, a string to be set with the error message if needed.
    - **Output:** Returns True or False.
    - **Description:** Check if user enters a username that is valid in server's account list and then if the entered password match the password in file.
- ***Register module:***
  - **`int validateRegisterData(char *fullName, char *username, char *email, char *password, char *question, char *answer, char *errorMessage)`**
    - **Input:** user's full name, username, password, email, and security question and answer entered by user, a string to be set with the error message
    - **Output:** Returns int userID.
    - **Description:** Checks to see if user inputs valid info when creating an account and creates new account if data is valid.
- ***ResetPassword module:***
  - **`void validReset(int userID ,const char *currentpw, const char *newpw, const char *Squestion, const char *Answer, char *errorMessage)`**
    - **Input:** the userID of the client, current password, new password, security question, and place to store error Message.
    - **Output**: returns true if the user has proven his credentials
    - **Description**: If the provided information matches with the one saved in the database, it allows user to change password.
- ***AddFriend module:***
  - **`bool SearchFriend(const char *Friend_Username, char *errorMessage)`**
    - **Input:** The name of the friend, location to store error Message
    - **Output:** Returns True or False
    - **Description:** Searches within the database the username inputted by user.

    - **`void SendingFriendRequest(int userID, int friendID, char *errorMessage)`**
        - **Input:** the id of the person sending the request, the id of the friend, location to store error Message.
        - **Description:** Sends a friend request to a contact specified by the user.

- ***FriendRequest module:***
    - **`void getRequests (int userID, int clientSocketFD)`**
        - **Input:** the id of the client, client's socket descriptor
        - **Description:** get the requests list of a user.
    - **`void ApproveRequest(int userID, int friendID, bool accepted)`**
        - **Input:** the id of the user who sent the friend request, and a boolean to specify if the client accepted the request or declined.
        - **Description:** When friend request is received, the user need to accept or decline. If accepted, the user can chat with the other user. If decline, other user will received a notice. This function updates the friends requests list and contacts list also.
- ***Home module:***
    - **`void returnRequests(char *serverMessage, int userID)`**
        - **Input:** id of user,
        - **Description:** returns the number of friends requests the user has.
    - **`void returnNumberFriends(char *serverMessage, int userID)`**
        - **Input:** id of user,
        - **Description:** returns the number of friends the user has.
    - **`void returnContacts(char *serverMessage , int userID, int pos)`**
        - **Input:**
        - **Description:**
- ***Chat module:***
    - **`void returnChat( int user1, int user2, int lastUpdate, int clientSocketFD)`**
        - **Input:** the ID's of the users of this chat and the last update got by the client.
        - **Description:** send the chat history
    - **`void saveMessage(char *m,int user1, int user2, int senderID)`**
        - **Input:** the sent message, first user ID, second user ID, sender's ID

        - **Description:** update the chat history
- **RecoverPassword module:**
    - **`bool validRecover(const char *username,const char *Squestion, const char *Answer ,char *errorMessage)`**
        - **Input:** username, security question, answer, and location to store error Message
        - **Output:** returns true or false
        - **Description:** If the provided information matches with the one saved in the database, it allows user to recover password.
- **DataStruct module:**
    - **`void addMessageToChat(message m, chat c)`**
        - **Input:** the message m to be added to the chat c
        - **Description:** Adds the message into the chat log
    - **`void addFriend(int friendID, int userID)`**
        - **Input:** the ID of the friend to be added, the user ID
        - **Description:** Stores the friend into their personal friend list, both for user and friend.
    - **`void addRequest(int userID, int friendID)`**
        - **Input:** the user ID, the friend ID who is sending the request
        - **Description:** Stores the friend request from friend to user's friend request list
    - **`int getStatus(int userID)`**
        - **Input:** user ID
        - **Output:** the status of the user
        - **Description:** fetches the status the user
    - **`void setStatus(int userID, int stat)`**
        - **Input:** user ID, the status
        - **Description:** Sets the status of the user
    - **`int newAccount(char *fullName, char *username, char *email, char *password, int question, char *ans)`**
        - **Input:** full name of the registree, username, email, password, security question and the registree's answer to it
        - **Output:** returns the index of the registree with respect to the accounts list.
        - **Description:** Stores the information of the registree into the accounts list.
    - **`int newChat(int user1, int user2)`**
        - **Input:** ID of first user in chat, ID of second user in chat
        - **Output:** returns the index of the chat with respect to the chat list.

- **Description:** Initializes a new chat and saves the users in the chat

1.4. Overall Program Control Flow

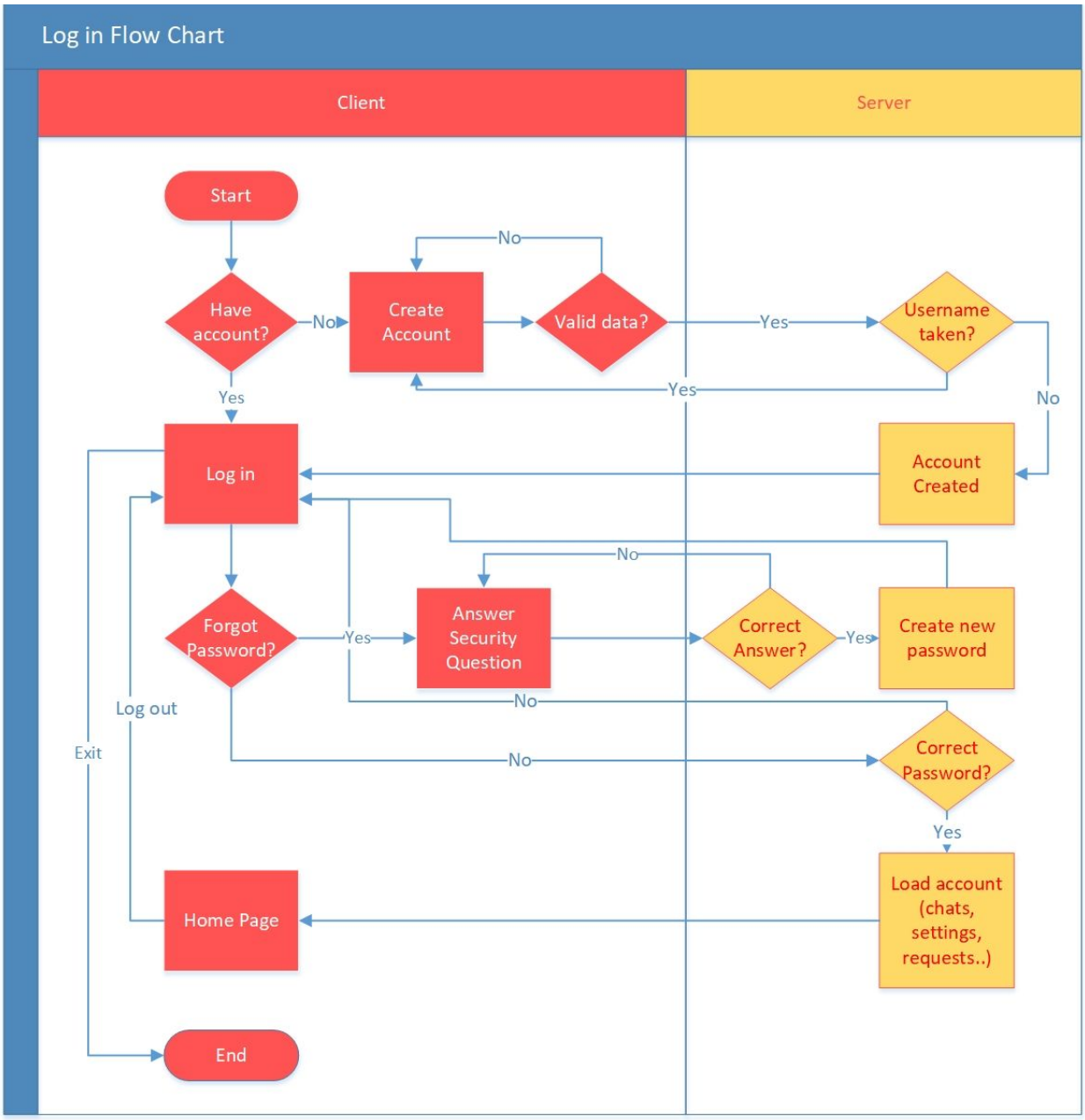

Figure 2 - login flow chart

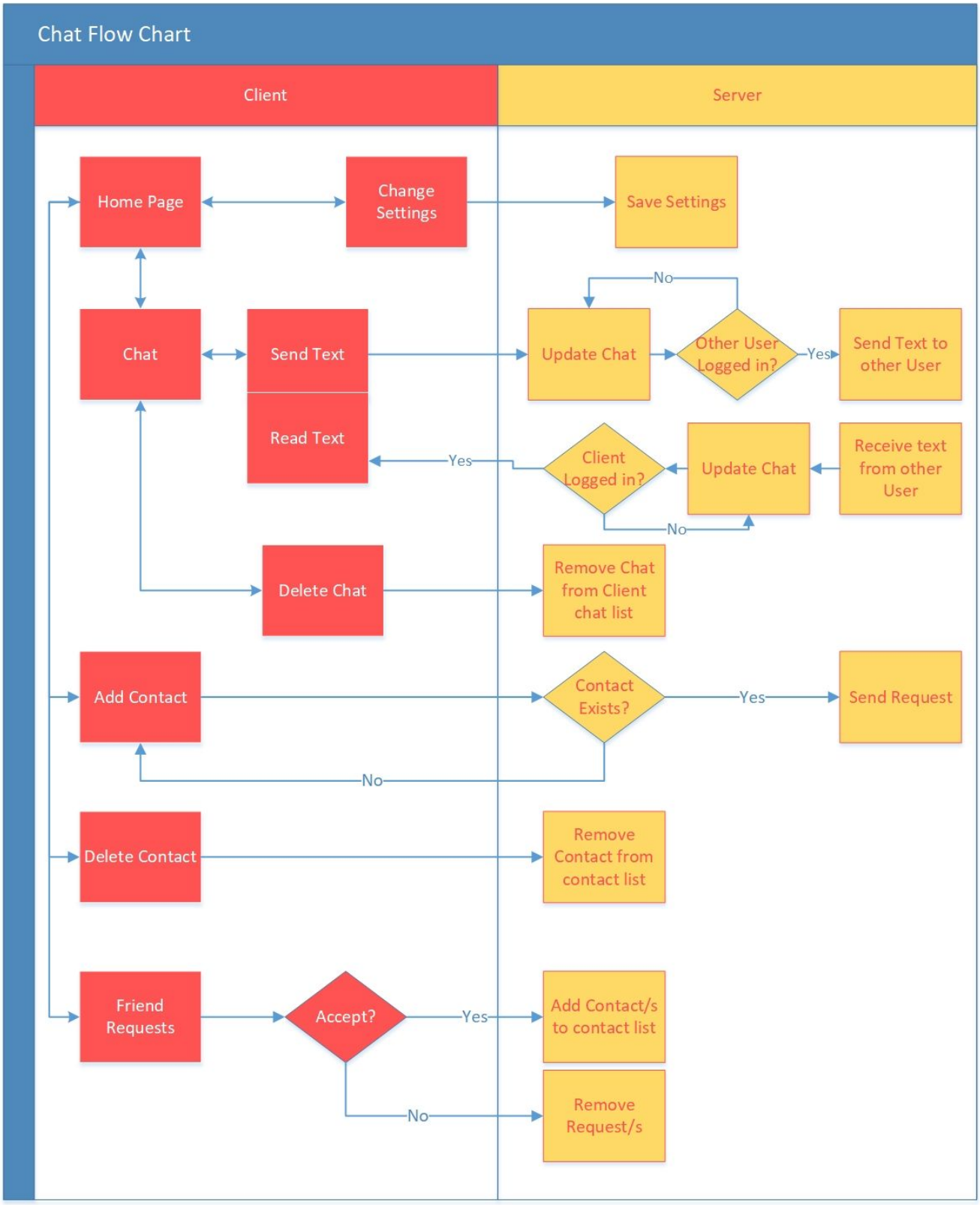

Figure 3 - chat flow chart

2. Client Software Architecture Overview

2.1. Main Data Types and Structures

- Structures:
  - **`friend`** - contains the username, ID, and the status of a friend.
  - **`message`** - contains the username of the sender, the contents , and the id of the message.
  - **`Login -`** contains the widgets to be used for login page.
  - **`Reset -`** contains the widgets to be used for password reset page.
  - **`Recover`** - contains the widgets to be used for password recover page.
  - **`Registration -`** contains the widgets to be used for registration page.
  - **`Chat`** - contains the array of users in the chat, history of the chat, and length of chat
- Data types
  - **`widgets`** - a widget for every button, box, inputs, ... used in the program.
  - **`Exit`** - exits out of the entire program if flag is set
  - **`userIDGlobal`** - stores the value of user's ID to be globally
  - **`usernameGlobal`** - saves the username to be used globally

2.2. Major Software Component

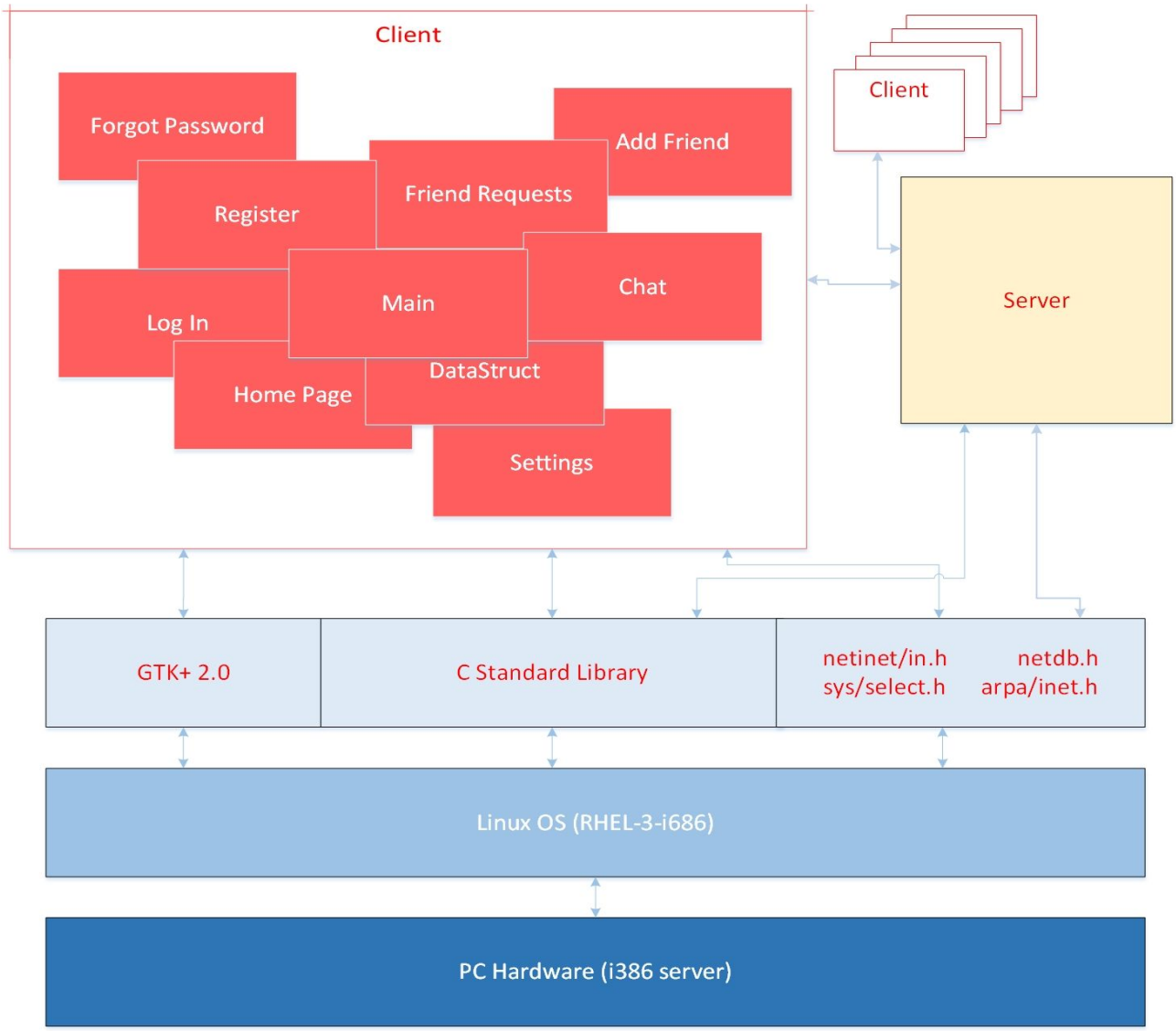

Figure 4 - client software components

The client has a main function, which calls all other functions from the modules. When a message is received by the server, a proper function is called with the corresponding parameters given in the message. Then, it prints the gtk window with the data given by the server.
When the client needs some data, its main function write a message to the server to request this data.

2.3. Module interfaces

- ***Login module:***
  - `void printLogin(GtkWidget *widget, GtkWidget *list[])`
    - **Input:** widget is the button clicked to call this function, list is the needed widgets to print the new page.

        - **Description**: Prints the display and buttons of the login page.
    - **`void getLoginData(GtkWidget *widget, login *data)`**
        - **Input:** a pointer to a widget**,** login information inputted by the user.
        - **Description**: Fetches the username and password provided by the user.
    - **`int validateLoginData(const char *username, const char *password)`**
        - **Input:** username and password
        - **Output:** Returns an integer corresponding to the user's ID or -1.
        - **Description**: Sends the login data to the server, and gets the answer back from the server. Returns the user ID if data is valid or -1 if invalid.
- ***Register module:***
    - **`void PrintRegister(GtkWidget *widget,  GtkWidget *list[])`**
        - **Input:** widget is the button clicked to call this function, list is the needed widgets to print the new page.
        - **Description**: Prints the display and buttons of the register page.
    - **`void getRegisterData(GtkWidget *widget, registration *data)`**
        - **Input:** a pointer to a widget, registration information inputted by the user
        - **Description**: Fetches the full name, username, password, security question and answers, and email address provided by the user. Check before sending to server
    - **`int validateRegisterData(const char *fName, const char *lName, const char *uName, const char *email,const char *password)`**
        - **Input:** first and last name of the user, username, email, and password
        - **Output:** an integer corresponding to the user's ID
        - **Description**: Determines whether user inputs meet the minimum requirements. If so, it sends to the server to be saved.
- ***FriendsRequest module:***
    - **`void getRequests(int userID)`**
        - **Input:** ID of the user
        - **Description**: This function will retrieve a list of friend requests from the server
    - **`void printFriendRequest(GtkWidget *widget, int userID)`**

        - **Input:** widget is the button clicked to call this function, ID of the user
        - **Description**: Prints the display and buttons of the friends request page.
    - **`void acceptRequest(GtkWidget *widget, int data[])`**
        - **Input:** a pointer to a widget ,an integer type array stores user IDs.
        - **Description**: Writes to the server that a user has accepted a friend request.
    - **`void declineRequest(GtkWidget *widget, int data[])`**
        - **Input:** a pointer to a widget ,an integer type array stores user IDs.
        - **Description**: Writes to the server that a user has declined a friend request.
- ***Addfriend module:***
    - **`void printAddFriend(GtkWidget *widget, GtkWidget *list[])`**
        - **Input:** widget is the button clicked to call this function, list is the needed widgets to print the new page.
        - **Description**: prints the add friend page
    - **`void getAddFriendData(GtkWidget *widget, add *data)`**
        - **Input:** widget is the button clicked to call this function, data inputted the user
        - **Description**: gets the user name when the button is clicked
    - **`int searchFriend(const char *f)`**
        - **Input:** a pointer to character type corresponding to the friend's username
        - **Output:** an integer corresponding to the friend's ID if found, -1 otherwise
        - **Description**: send the username to the server, returns the id of the user if found, -1 otherwise
    - **`void addFriend(GtkWidget *widget, login *data);`**
        - **Input:** a pointer to a widget, data inputted by the user when logging in
        - **Description:** sends the friendID to the server to send friend request
- ***Chat module:***
    - **`void PrintChat(GtkWidget *widget, int data[])`**
        - **Input:** widget is the button clicked to call this function, data of the user
        - **Description**: Prints the display and buttons of the chat page.
    - **`void getChat(int user1, int user2)`**

        - **Input:** an integer corresponding to the first user's ID, an integer type corresponding to the second user's ID
        - **Description**: Gets the chat history from the server
    - **`void sendMessage(GtkWidget *widget, int data[])`**
        - **Input:** widget is the button clicked to call this function, list is the needed widgets to print the new page.
        - **Description**: Sends the message to the server. Server fulfills request by sending the message to the friend listed under ID.
- ***Home module:***
    - **`void getContacts(friend *f, int pos, int userID)`**
        - **Input:** a friend type array, an integer corresponding to the position of the friend, an integer corresponding to the user's ID
        - **Description:** This function will fetch the contacts list from the server and display it on the client side specific to each individual client
    - **`void printHome(GtkWidget *widget, int userID)`**
        - **Input:** widget is the button clicked to call this function, ID of user
        - **Description:** Prints the display and buttons of the home page.
    - **`void getNumberFriends(int *fCount, int userID)`**
        - **Input:** the number of friends, ID of the user
        - **Description:** Gets the number of friends from the server
    - **`void getNumberRequests(int *rCount, int userID)`**
        - **Input:** the number of friend requests, ID of the user
        - **Description:** Gets the number of friend requests from the user
    -
- ***ResetPassword module:***
    - **`void printResetPassword(GtkWidget *widget, GtkWidget *list[])`**
        - **Input:** widget is the button clicked to call this function, list is the needed widgets to print the new page.
        - **Description:** prints the gtk page of reset password
    - **`void  getResetData(GtkWidget *widget, reset *d)`**
        - **Input:** widget is the button clicked to call this function, data inputted by the user
        - **Description:** fetches the current password, new password, security question, and answer from the user.
    - **`void validateResetData(const char *currentPW, const char *newPW, int question, const char *answer)`**

        - **Input:** current password, new password, security question, and answer
        - **Description:** checks to see whether user has the credentials to change password

- **RecoverPassword module:**
    - **`void printRecoverPassword(GtkWidget *widget, GtkWidget *list[])`**
        - **Input:** widget is the button clicked to call this function, list is the needed widgets to print the new page.
        - **Description:** prints the gtk page of recover password
    - **`void  getRecoverData(GtkWidget *widget, recover *d)`**
        - **Input:** widget is the button clicked to call this function, data inputted by the user
        - **Description:** fetches the username, security question, and answer
    - **`char *validateRecoverData(const char *username, const char *answer, int question, char *password)`**
        - **Input:** username, answer, question, and location to store password.
        - **Output:** returns the password or null
        - **Description:** checks to see whether user has credentials to recover password. If so, it reveals the password to the user.

2.4. Overall Program Control Flow

Please refer to figure 2 and figure 3 in section **1.4** for the overall program control flow.

3. Installation

3.1. System requirements, Compatibility

3.1.1. Linux OS 2.6.32-696.18.7.el6.x86_64

3.1.2. PC(x86_64 Server)

3.2. Setup and Configuration

No setup or configuration is needed.

3.3. Building, Compilation, and Installation

The Following steps are required for proper installation:

1. On the terminal window, extract file using the following command:
    - "`gtar xvzf Chat_Alpha_src.tar.gz`"
2. Compile the chat application by entering the following command:
    - type "`make`" to compile the chat and the server.
3. Once compilation is done, enter:

- “`./bin/chat zuma 7777`” to start the chat application.
- “`./bin/server 7777`” to start the server.
- NB: If the port 7777 was taken, please replace 7777 with any 4 digit number greater than 2000 for server and chat port.

4. Documentation of Packages, Modules, and Interfaces

4.1. Data Structures

4.1.1. Server data structures

- **clientAccount**
  - **`char *fullName` -** pointer to the client’s full name
  - **`char *username`** - pointer to the client’s username
  - **`int ID`** - stores the value of the client’s ID
  - **`char *email`** - pointer to the client’s email address
  - **`char *password`** - pointer to the client’s password
  - **`char *question`** - pointer to the client’s security question
  - **`char *answer` -** pointer to the client’s answer to the security question
  - **`int font` -** stores the value of client’s font preference
  - **`int color` -** stores the value of client’s color preference
  - **`account *friends`** - pointer to the client’s friends list
  - **`account *requests` -** pointer to the client’s friends request list
  - **Code:**

```
typedef struct clientAccount{
     char *fullName;
     char *username;
     int   ID;
     char *email;
     char *password;
     char *question;
     char *answer;
     int   font;
     int   color;
     account *friends;
     account *requests;
}account;
```

- **Mes**
  - **`account sender` -** contains the information of the sender
  - **`char *s text` -** pointer to the text sent by the sender
  - **Code:**

```
typedef struct mes{
```

```
        account sender;
        char *s text;
    }message;
```

- **Chat**
  - **`int users[2]` -** contains ID of the users in the chat (max of 2 users at the moment)
  - **`message *history`** - pointer to the chat history
  - **`length`** - stores the length of the chat
  - **Code:**

```
typedef struct Chat{
    int users[2];
    message *history;
    int length;
}chat;
```

4.1.2 Client data structures

- **Friend**
  - **`int id` -** stores the ID of the friend
  - **`char username[26]`** - friend's username
  - **`int status`** - stores the value of the friend's status (online, idle, offline)
  - **Code:**

```
typedef struct Friend{

    int id;
    char username[26];
    int status; //online or offline
}friend;
```

- **message**
  - **`int senderID` -** stores the value of the sender's ID
  - **`char *text`** - pointer to the message sent by the sender
  - **`int messageID` -** stores the value of the message ID
  - **Code:**

```
typedef struct m{

    int senderID;
    char *text;
    int messageID;
}message;
```

4.2. Functions and Parameters

4.2.1. Client functions

- **`void printAddFriend(GtkWidget *widget, GtkWidget *list[]) -`** prints the AddFriend gtk page.
- **`void getAddFriendData(GtkWidget *widget, add *data -`** When the user clicks search, gets the username from the search input box, sends it to the server, and prints a button to send a friend request to this user. If the username has not been found in the database, an error message will be printed.
- **`int searchFriend(char *f) -`** called inside the "getAddFriendData" function. It sends a request to the server to check if the username is found in the database. If found, It returns the id of the friend. It returns -1 otherwise.
- **`void addFriend(GtkWidget *widget, login *data) -`** sends a friend request to the corresponding user, if the client hit "Send Request" button.
- **`void getChat(int user1, int user2) -`** gets the chat history of a particular chat.
- **`void PrintChat(GtkWidget *widget, int data[]) -`** prints the chat page with the old chat history.
- **`void sendMessage(GtkWidget *widget, int data[]) -`** When the client clicks send, this function updates the chat, sends the message to the server to update chat history and prints the updated chat.
- **`void getRequests(int userID)-`** gathers all the data of incoming friends requests which is used to create "visual" requests.
- **`void printFriendRequest(GtkWidget *widget, int userID)`** - creates an updated window with the current friend requests. Displays the number of requests, the ID of the person, and a button Yes/No to accepts or not.
- **`void acceptRequest(GtkWidget *widget, int data[])-`** Writes to the server that a user has accepted a friend request.
- **`void declineRequest(GtkWidget *widget, int data[]))`**-Writes to the server that a user has declined a friend request.
- **`void getContacts(friend *f, int pos, int userID)`**- gathers number of contacts and relative data such as ID and info stored on the server
- **`void printHome(GtkWidget *widget, int userID)`**- creates home page widows with the updated list of contacts, relative buttons to open or delete chats plus buttons to open the 'friend requests' and 'settings' pages.
- **`void getNumberFriends(int *fCount, int userID)-`** Gets the number of friends from the server

- **`void getNumberRequests(int *rCount, int userID)-`** Gets the number of friend requests from the user
- **`void printLogin(GtkWidget *widget, GtkWidget *list[])-`** creates the Login window which displays text boxes for identification and links to create a new password and create an account.
- **`void getLoginData(GtkWidget *widget, login *data)`** - gets the login credentials. Prints error if not valid. Opens the home page window if valid.
- **`int validateLoginData(const char *username, const char *password)`** - checks if login credential are valid though the server. Creates error message if not valid.
- **`int main(int argc, char *argv[])`** - takes care of all the communication between the client and the server. Data is sent and received here before being forwarded to the recipient.
- **`void PrintRegister(GtkWidget *widget, GtkWidget *list[])`** - creates the widow for registration with text boxes to input the necessary info to create an account.
- **`void getRegisterData(GtkWidget *widget, registration *data)`** - gets all the data required to create an account. Redirect to home page if all the inputs are valid or prints error if not valid.
- **`int validateRegisterData(const char *fName, const char *lName, const char *uName, const char *email,const char *password)`** - checks if the inputs used to create a new account are valid. Creates an error message if not.
- **`void printResetPassword(GtkWidget *widget, GtkWidget *list[])`** - creates window to change password which includes the security question that depends on the ID and a text box to input the answer
- **`void getResetData(GtkWidget *widget, reset *d)-`** gets all the data required to reset password.
- **`char *validateRecoverData(const char *username, const char *answer, int question, char *password)-`** checks if reset credentials are valid through the server. Prints an error message if not.
- **`void PrintSettings(GtkWidget *widget, GtkWidget *list[])-`** creates the settings windows
- **`void printRecoverPassword(GtkWidget *widget, GtkWidget *list[])-`** prints the gtk page of recover password
- **`void getRecoverData(GtkWidget *widget, recover *d)-`** fetches the username, security question, and answer
- **`char *validateRecoverData(const char *username, const char *answer, int question, char *password) -`** checks to see whether user has credentials to recover password. If so, it reveals the password to the user.

4.2.2 server functions:

- **bool validLogin(char *username, char *password, char *errorMessage)-** Check if user enters a username that is valid in server's account list and then if the entered password match the password in file.
- **int validateRegisterData(char *fullName, char *username, char *email, char *password, char *question, char *answer, char *errorMessage)-** Checks to see if user inputs valid info when creating an account and creates new account if data is valid.
- **bool SearchFriend(const char *Friend_Username, char *errorMessage)-** Searches within the database the username inputted by user.
- **bool searchByFullName(const char *fullName)-** Searches within the database for a result that matches a fullname passed in to the funntion. It This function returns true if the name is already registered in the database otherwise, returns false.
- **void SendingFriendRequest(int userID, int friendID, char *errorMessage)-** Sends a friend request to a contact specified by the user.
- **void getRequests (int userID, int clientSocketFD)-** get the requests list of a user.
- **void ApproveRequest(int userID, int friendID, bool accepted)-** When friend request is received, the user need to accept or decline. If accepted, the user can chat with the other user. If decline, other user will received a notice. This function updates the friends requests list and contacts list also.
- **void UpdateHome(int contactsID[], char *username[], bool status[], int userID)-** get the contacts list of a user.
- **void returnRequests(char *serverMessage, int userID)-** return the number of friend requests of a user.
- **void returnNumberFriends(char *serverMessage, int userID)-** return the number of friend of a user.
- **void returnContacts(char *serverMessage , int userID, int pos) -** return one contact from the list of contacts of the user.
- **void returnChat(int user1, int user2, int lastUpdate, int clientSocketFD)-** returns chat history.
- **void saveMessage(char *m,int user1, int user2, int senderID)-** saves a message in the chat history.
- **void SendMessage(char *message)-** return a message requested by a user.
- **bool validRecover(const char *username,const char *Squestion, const char *Answer ,char *errorMessage)-** If

the provided information matches with the one saved in the database, it allows user to recover password.

- **`void addMessageToChat(message m, chat c)-`** Adds the message into the chat log
- **`void addFriend(int friendID, int userID)-`** Stores the friend into their personal friend list, both for user and friend.
- **`void addRequest(int userID, int friendID)-`** Stores the friend request from friend to user's friend request list
- **`int getStatus(int userID)-`** fetches the status the user
- **`void setStatus(int userID, int stat)-`** Sets the status of the user
- **`int newAccount(char *fullName, char *username, char *email, char *password, int question, char *ans)-`** Stores the information of the registree into the accounts list.
- **`int newChat(int user1, int user2)-`** Initializes a new chat and saves the users in the chat

4.3. Communication Protocol

4.3.1. Explanation of communication protocol

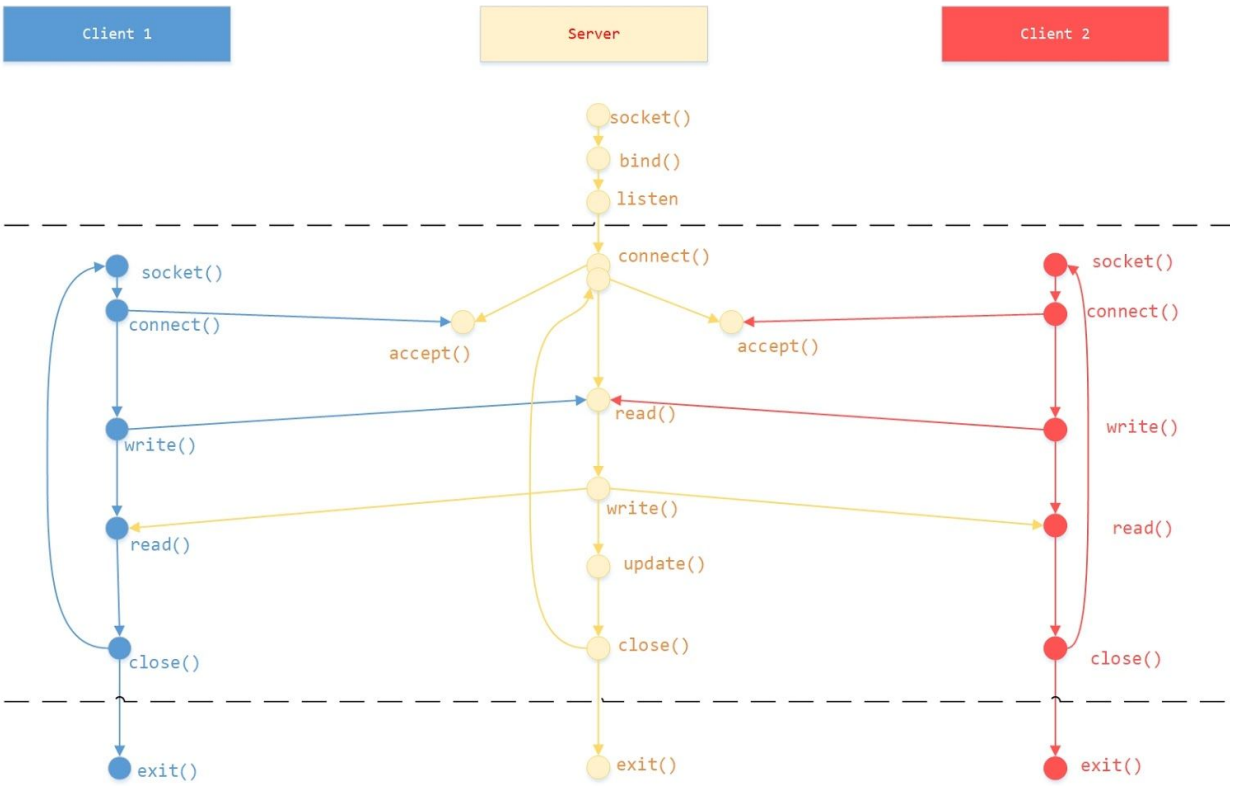

Figure 6. Client(s)- Server Communication

4.3.2. Explanation of communication related function calls

- You can refer to figure 2 and figure 3 in section 1.4 for a better understanding of the communication related function calls.

- What we mean in the following description by "call" is that the client main function will be calling the proper function, which will send a message to the server. The server will read this message, and call the proper function depending on the received message. The server's function will send back a message to the client. Then, the client will take a decision depending on the message.

- **Login module:**
  - When the `"validateLoginData"` function from the client side is called, it calls the `"validLogin"` function from the server side to check if the given data is found in the database and return the answer(found or not) and an error message if needed.
- **Register module:**
  - When the `"validateRegisterData"` function from the client side is called, it calls the `"addAccount"` function from the server

side to check if the given data is valid. If the data is valid, it creates the client profile, othewise, server sends an error message back to to the client side.

- **Home module:**
  - The "`getContacts`" function calls the "`returncontacts`" function to get the user's contact list. The "`getNumberFriends`" function calls the "`returnNumberFriends`" function to get the number of friends in friend list. The "`getNumberRequests`" calls "`returnRequests`" to get the number of requests in friend request list.
- **Chat module:**
  - The 2 client functions "`getChat`", and "`sendMessage`" call the 2 server functions "`sendChat`", and "`sendMessage`" respectively to make any updates to the chat.
- **FriendRequest module:**
  - The client function "`getRequest`" call the server's functions "`getRequests`" to get the list of requests. "acceptRequest" and "declineRequest" calls the server function "ApproveRequest" to add the friend in the friend list if accepted or ignore if not.
- **AddFriend module:**
  - The 2 client's functions "`searchFriend`" and "`addFriend`" will call the 2 server's functions "`SearchContact`" and "`SendingFriendRequest`" respectively to search the friend in the database and send a friend request.
- **ResetPassword module:**
  - The client function "`validateResetData`" will call the server function "`validReset`" to check if the given data is found in the database and return the answer(found or not) and an error message if needed.
- **RecoverPassword module:**
  - The client function "`validateRecoverData`" will call the server function "`validRecover`" to check if the given data is found in the database and return the answer(found or not) and an error message if needed.

5. Development Plan and Timeline

5.1. Timeline

5.1.1. By End of Week 8 (Alpha version):

- All the client modules (gtk part).

5.1.2. By End of Week 9 (Beta version):

5.1.2.1. All the server modules.

5.1.2.2. The communication between server and client (client and server main function).

5.1.3. By End of Week 10 (Final version):

5.1.3.1. Testing

5.1.3.2. Extra credit features

5.2. Partitioning of Tasks and team members responsibilities:

- **Daniel:** client and server Login modules
- **Ragui**: client and server chat and main modules
- **Hoon**: client and server FriendRequest modules
- **Jacobis**: Makefile, client and server Register modules
- **Martino**: client and server Home and ResetPassword modules
- **Hanqi**: client and server AddFriend and settings module

5.3. Team Responsibilities (all the team):

5.3.1. Data structures modules

5.3.2. Debugging and testing

5.3.3. Maintenance of software and user manual

6. Backmatter

6.1. Copyright:

6.2. References:

6.2.1. http://www.linuxhowtos.org/C_C++/socket.htm

6.2.2. http://www.linuxhowtos.org/data/6/client.c

6.2.3. http://www.linuxhowtos.org/data/6/server.c

6.2.4. https://developer.gnome.org/gtk-tutorial/stable/book1.html

6.3. Index:

Accounts List..... 4
addFriend..... 5
ApproveRequest..... 5
Chat List..... 4
Client..... 9
DeleteAll..... 13
DeleteChatHistory..... 6
DeleteMessage..... 6
Friend..... 9
getAddFriendData..... 12
getAnswer..... 12
getChat..... 13
getColor..... 11
getContacts..... 13
getFont..... 11
getFriendRequests..... 12
getLoginData..... 10
getNewPass..... 15
getPassword..... 11
getQuestionAnswer..... 14
GetRegisterData..... 10
getSecurityQuestion..... 14
getSettings..... 11
getUsernameReset..... 14
Message..... 4
printAddFriend..... 12
printChat..... 13
printFriendRequest..... 12
printLogin..... 10
printNewPass..... 15
printQuestion..... 14
PrintRegister..... 10
printResetPassword..... 14
PrintSettings..... 11
SearchContact..... 5
searchFriend..... 12
SendMessage..... 6
SendRequest..... 5
Server..... 4
setAnswer..... 12
Setup..... 15
updateColor..... 11
UpdateColor..... 6
updateFont..... 11
UpdateFont..... 6
updatePassword..... 11
UpdatePW..... 6
validateLoginData..... 10
validLogin..... 5
ValidRegister..... 5