Design and Implementation

Server Side

1. The main code is in `server.cpp`.
2. `lockless_queue.cpp` implements a lockless queue so that the requests in various threads can be answered one by one.
3. `utils.cpp` contains functions which are being used for parsing requests and generating properly formatted request.

The format of the request follows the following format:

/command:associated_username:various_parameters(such as message etc.)#

At the server side, we have a database which stores everything. The passwords are encrypted before storing and a salt is being used.

For login, there are two ways:
1. One can directly login using their account on the LDAP server
2. They can register and their details will be stored in the database

We have the following four tables:
1. Friends - Stores the relationship between different users
2. Chats - Stores chat_ids and chat_name(for further extension to group chat)
3. Users - Stores details of all users including last_seen time and whether they are online or not
4. Messages - This table stores all messages corresponding to all chats. There is a many to one relationship with chats
5. Chats_users_xref - A table made to get a many-to-many relationship with chats(Generalised so that it can be extended easily for group chat)

To compile the server.cpp file just run `make`.

It has the following dependencies:
1. Libsodium - https://download.libsodium.org/doc/installation/
2. Libsqlite3 - sudo apt-get install libsqlite3-dev
3. Libldap - sudo apt-get install libldap2-dev

The server maintains a queue of all the requests it receives and then writes to the proper socket depending on the request.

Raw SQL queries are being used to extract data from the database.

To prevent overflow of stack, data is not being stored in data structures on the stack and being saved directly to the database on every request. Everything is happening over TCP sockets.

Android

On android we maintain a Database class which has information corresponding to the user using the app.

A Person class stores information, chats and the relationship with the user.

Every message goes through NetworkService which parses the request. Network Service spawns two threads - Receive Thread and Sending thread. Receive thread has the whole control over the database. The service keeps the socket open in the background to keep a persistent connection.

Web Client

We use websockify library(https://github.com/novnc/websockify) to get javascript talk to raw sockets.

This is a simple implementation implementing chat with only online users. Currently has some bugs causing page to refresh when it should not.

The websockify client needs to be run as a proxy(see the above github link) for relaying communication to the server.