CN Project Report

AISHNA AGRAWAL | ADITYA PUSALKAR | JANVI THAKKAR

Mini-WhatsApp

Overview:

A simple networking tool which has many functionalities which are similar to the popular chat app: **Whatsapp**.

Goals:

Design and implement a Mini-WhatsApp tool which supports features such as chatting, file sharing, and profile creation.

Design Structure:

Following are the various features and implementations of the project.

Initialization:

To download the Mini-Whatsapp github repository, we need to clone it using the following command:

git clone https://github.com/aishna-agrawal/MiniWhatsApp

After downloading it we follow these steps:

To set up our Mini-Whatsapp, we first need to start up the main server (registry.py). Simply run the following command:

$ python registry.py

Now the user can start using Mini-Whatsapp by starting the chat program (peer.py) using the following command.

$ python peer.py

Note: The peer connects to the registry using its IP address which will be displayed on the registry side. Also it can be hard-coded if convenient.

Main features:

There are many features that have been packed into this application. Namely:

1. **Account Creation:** The user can make an account to authorize access to the application. Without creating an account and logging in, you cannot access the features of Mini-Whatsapp
2. **Login/Logout:** The user can login to the application and access the various features. No unauthorized access to account details of the user to other users, such as username, password and status.
3. **Security:** The user has to provide a password to login to the application. The password is masked to provide security to users. Also the username is unique and cannot be overwritten.
4. **Database:** Using MongoDB database to store essential data like account details and currently online users.
5. **File Sharing:** Users can share any text file (locally saved) to the user that he/she is currently chatting with.
6. **Group Chat:** Users can simultaneously chat with multiple users at the same time with the group chat feature.
7. **Status:** The user can have a status (text description) in their profile, which is modifiable and can be viewed by other users
8. **Search:** The user can search for a user by their username. It will notify the user if the account is online (returns IP address), offline or does not exist.
9. **Chat Saving:** The user can request to save the chat while connecting to the other user.

Menu:

The Menu will be displayed once the peer is connected successfully to the client. There will be 8 options provided. Press the required number to execute the command.

1. **Create Account:** Create an account to use Mini-Whatsapp. You will have to provide a unique username and a password.
2. **Login**: Log into Mini-Whatsapp using the account created previously. You will have to provide a username and password to login. You also need to specify the port that you should use to start the peer client (More details in the 'Communication and Connection Maintenance' section). Password is masked for the purpose of security.
3. **Logout:** Logout from your current session of Mini-Whatsapp. You will then exit the application.
4. **Search:** Search for a particular user using their username.
5. **Start Chat:** Using the username of a user you can send a request to the user to start a chat session with them. On the receiver-side the user will receive a chat request pop-up where they must type "**OK**" to start the chat. There is also another option of "**OK-SAVE**" which the user can choose to save the chat messages.
6. **Change Status:** The user can change their status using this command. The user will have a default status (a description in text). He/She can choose to change the status if he/she wishes to do so.
7. **See Status:** The user can view the status of any user by entering the username of the user.
8. **Start Group Chat:** The user can also create a group chat with any number of users. The port for the group chat server must also be specified. The user will be able to add the users by entering their names and sending requests to each user.

Database:

We are using **MongoDB** to store all the information about the peers which includes storing their username, password and status. Pymongo library is used to integrate between the MongoDB database and the script. Operations done on the database includes:

- **Registration of account** - entry is made in DB with username, password, status fields.
- **Account exist** - checks in the database whether the account with given username exist or not (maintains uniqueness and provides security)
- **Update status** - updates the status of the given username whenever he/she requests
- **Get-password** - function gets the password from the database and checks before any user tries to login.
- **Get-status:** Get the current status of particular username
- **Get-peer-ip-port:** Get the ip and port of particular username
- **User_login:** Log in the database, when the user logs in a new field online peers are created in DB that contains details about all the clients that are currently logged in.
- **User-logout** - function log-out the peer and removes its name from online peers.

Communication and Connection Maintenance:

1. We have the main server (registry) which is responsible for dealing with different clients. The connection between the registry and different clients is of **client-server** type.
2. The type of communication between the clients is **P2P**. The main server helps the peers (users) who login, to establish a connection between each peer.
3. Peers act as both server and client while sending and receiving messages. The Peers are started by executing the **peerMain** class. Then a **TCP** socket is created for receiving details from the registry and other clients.

4. A special **UDP** socket is also created for each peer. This socket sends an Acknowledgement message ("**ACK**") to the registry every second. The registry receives this message and gets notified that this client is still connected to it. Any short or abrupt connection loss will lead to registry checking for the ACK and wait for the ACKs to completely stop. This ensures that the peer remains connected.
5. The Chat feature uses the following logic. For each peer we have **2** ports at which we are listening. The first port is initiated as a receiver and the second as a sender. The first port is created when the user logs in. If **peerA** sends a message, the sender port sends it to the receiver port of **peerB** and vice versa. The sender port is closed after the single chat is closed. The receiver port is kept open for messages from registry or any incoming chat requests from other users, as and when needed. These two ports and their connected processes run simultaneously in the background by using a very important feature known as **threading**.
6. **Concurrent Server:** The Group Chat features a Concurrent group chat server. The server is created by the user requesting it on a custom port decided by the user. The server is then created and waits for incoming requests for chatting. The user can request other users to join the chat. Once the other users receive the request, they can join the chat by typing in "**OK-GROUP-port**" where (port) will be the port of the group chat server. The clients connect one by one and everyone will be notified of joining the chat room. Now everyone who has joined can send their messages to everyone. The messages will be **broadcasted** to every user present in the group using a TCP connection. The user can leave the chat by typing in "**QUIT**".
7. We can create multiple chat servers at the same time, by specifying different ports at the time of creating the chat room.

Other Key Features:

- File transfer can be done while the users are chatting with each other. Users can press "**:f**" in the terminal. After that we can enter the file name to send the file.
- When requesting to chat with a peer, the person receiving the request can type in "**OK-SAVE**" to save the chats that follow. The file is live-saved and under the name of the username who typed in the same.
- When the user wants to leave the chat (single chat), he/she can type in "**:q**" and leave the chat and return to the main menu.

Deployment/ Dependency List:

- The main requirement is the installation of MongoDb
- Its installation process can be seen here : link
- Once installed, run MongoDB compass and connect the database, so that you can analyse the database as the application is run.
- Some of the python libraries required are :
    - threading
    - pymongo
    - stdiomask
    - select
