Internet Technologies Lab Report
Assignment 2

Md Sahil
BCSE III
Roll-001710501029

1 Problem Statement

Write a multi-client chat application consisting of both client and server programs. In this chat application simultaneously several clients can communicate with each other. For this you need a single server program that clients connect to. The client programs send the chat text or image (input) to the server and then the server distributes that message (text or image) to all the other clients. Each client then displays the message sent to it by the server. The server should be able to handle several clients concurrently. It should work fine as clients come and go.

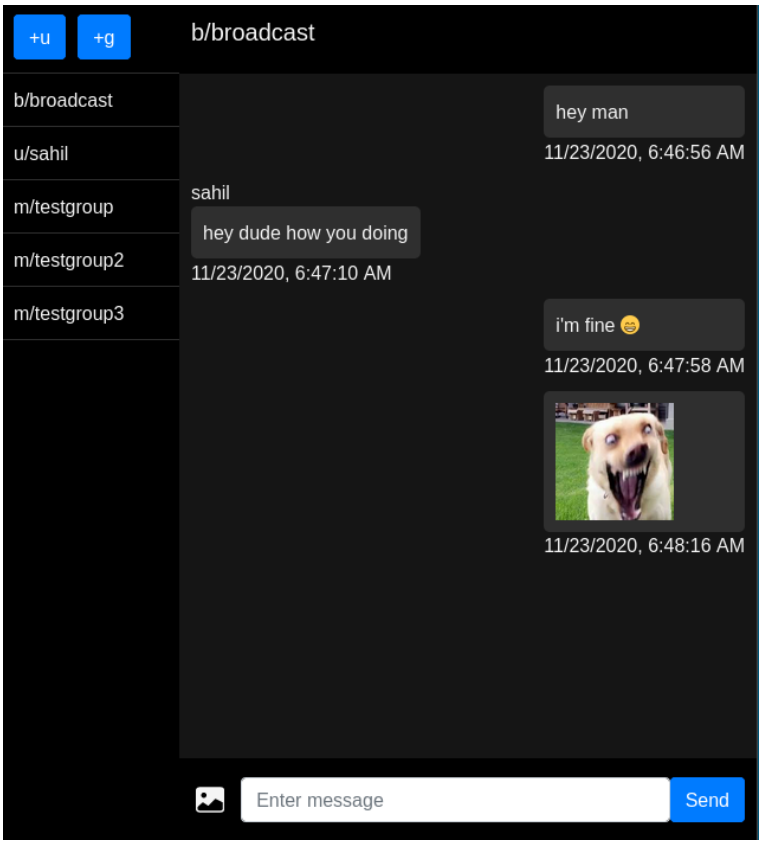
Develop the application using a framework based on Node.JS. How are messages handled concurrently? Which web application framework(s) did you follow? Prepare a detailed report of the experiments you have done, and your observations on the protocol layers thus created.

2 Development stack used

- Server:
  - Node js
  - Express js
  - Socket.io
- Client:
  - React js
  - Socket.io

3 Usage and Features

DingDing is an online real time chat application with support for images and text. Users can create accounts on the platform. User Id and password are stored for each user in the server. Users can connect with other users through their respective user Ids. Multicast groups can also be formed. A group consists of more than one users. There is also a broadcast group. Every user in the platform by default is present in that group and thus sending messages there reaches everyone in the platfrom.



4 Design & Implementation

4.1 Directory structure

```
.
|-- LICENSE
|-- README.md
|-- client
|   |-- build
|   |-- package-lock.json
|   |-- package.json
|   |-- public
|   |   |-- index.html
|   |   |-- manifest.json
|   |   `-- robots.txt
|   `-- src
|       |-- App.css
|       |-- App.js
|       |-- components
|       |   |-- ChatScreen.js
|       |   |-- ChatWindow.js
|       |   |-- LoginScreen.js
|       |   |-- MessageBox.js
|       |   `-- SidePanel.js
|       `-- index.js
|-- docs
|   `-- res
|       `-- ss.png
|-- package-lock.json
|-- package.json
`-- server
    |-- register.js
    |-- register_data.json
    `-- server.js
```

4.2 Framework

The server build on nodejs has two parts i.e. a Rest API part and a websocket part. Client data like user_id, password, list of connections, groups are stored in the server.

4.2.1 User data stored in server

Example user data stored in server:

```
"sahil": {
    "password": "test",
    "connections": {
        "broadcast": {
            "type": "b"
        },
        "bikram": {
            "type": "u"
        },
        "irin": {
            "type": "u"
        "Jucse": {
            "type": "m",
            "users": [
                "bikram",
                "irin",
                "sahil"
```

```
            ]
        },
        "memes": {
            "type": "m",
            "users": [
                "bikram",
                "sahil"
            ]
        }
    }
}
```

Example group data stored in the server

```
    "Jucse": {
        "type": "m",
        "users": [
            "bikram",
            "irin",
            "sahil"
        ]
    }
```

Here *type u* stands for unicast, *m* stands for multicast and *b* stands for broadcast. The broadcast group is present by default for each user.

4.2.2 The Rest API

The Rest API part of the server handles the following tasks:

- user signup
- user login and fetch user data
- add connection
- create new group

4.2.3 The Web Socket

The web socket part is used for the transfer of realtime messages. Messages are structured as json objects
Message structure:

```
{
    "contentType" : "..." // text or other media
    "type: : "...", // unicast, multicast, broadcast
    "body" : "...",
    "timestamp" : "...",
    "receiver" : "...",   // uid or gid or broadcast message
    "sender" : "...", // Id of user who wrote the message
}
```

When a message is sent from the client side, the server reads the message object, checks the sender and receiver and forwards the message to the correct user or group memebers accordingly.

In order to send image data images are converted into *base64* strings. The *contentType* field of the message json is set to *img* and the *base64 string* is set as the *body* field of the message json.