

CHAT APPLICATION

Ganadhish Acharekar
Akshat Shah
Aniket Roy
Arnav Shah
Saharsh Jain

ABSTRACT

Chat refers to the process of communicating, interacting and exchanging messages over the Internet. It involves two or more individuals that communicate through a service or software. Chat may be delivered through text, audio or video communication via the Internet. A chat application has basic two components, viz server and client. A server is a computer program or a device that provides functionality for other programs or devices. Clients who want to chat with each other connect to the server.

The chat application we are going to make will be both, like a chat room and a peer to peer chat. So this means that multiple users can connect to the chat server and send their messages. Every message is broadcasted to every connected chat user. Also, users can chat one to one.

CURRENT FEATURES:

- User Authentication
- Global chat room
- Public chat rooms on particular topics
- Peer to peer chat
- Storing chat history of peer to peer chats

TOOLS USED

1. HTML, CSS
2. Bootstrap CSS
3. JavaScript
 - A. Socket-IO Library
4. Python libraries and frameworks
 - A. Flask web framework
 - B. Flask SQLAlchemy
 - C. Flask Bcrypt
 - D. Flask Socket IO
 - E. Flask WTFForms
 - F. Flash Login

FILE STRUCTURE

```
project directory/  
----application package  
---- static/  
        ---- CSS files  
        ---- JavaScript files  
---- templates/  
---- HTML files  
---- __init__.py  
---- forms.py  
---- routes.py  
---- models.py  
run.py(file to be run)
```

REGISTRATION AND LOGIN

Registration and login is an important part of this application. A user first has to register by providing username, email id, password. This is then checked and authenticated before storing this information in the database to represent a user. Once registration is done, user can proceed to the login page and enter the necessary details to log in. The database is queried to check if given username and password are present. If yes, user proceeds to the chat page to begin chatting.

Flask WTFORMS has been used for creating registration and login forms and validating user inputs. For security, Flask Bcrypt has been used to hash passwords before storing them in the User database. Flask Login has been used for implementing user authentication system.

```
class RegistrationForm(FlaskForm):
    username = StringField('Username',
                           validators=[DataRequired(), Length(min=2, max=20)])
    email = StringField('Email',
                       validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    confirm_password = PasswordField('Confirm Password',
                                     validators=[DataRequired(),
                                     EqualTo('password')])
    submit = SubmitField('Sign Up')

    def validate_username(self, username):
        user = User.query.filter_by(username=username.data).first()
        if user:
            raise ValidationError('That username is taken. Please choose a
different one.')

    def validate_email(self, email):
        user = User.query.filter_by(email=email.data).first()
        if user:
            raise ValidationError('That email is taken. Please choose a
different one.')
```

DATABASE

A database is used to store the Users registering in the system. Data fields like username, email id, password are stored.

Another database is used to store the messages or chat history to be displayed for one to one communication. It contains fields like sender, receiver, timestamp, room name, data.

Thus, flask-sqlalchemy is used to make and query databases for future use.

```
class Rooms(db.Model):
    __bind__ = 'rooms'
    id = db.Column(db.Integer, primary_key=True)
    roomname = db.Column(db.String(250), unique=True)
    message = db.Column(db.String(50000), default='{"0":""}')
    count = db.Column(db.Integer, default=0)

    def __repr__(self):
        return "{'ID:{},'roomname:{},'count:{}}".format(self.id,
self.roomname, self.count)
```

SOCKETS

Sockets allow communication between two different processes on the same or different machines. A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the tcp layer can identify the application that the data is destined to be sent to. Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request. On the client side: The client knows the host name of the machine of which the server is running and the port number in which the server is listening.

To make a connection request-

The client also needs to identify itself to the server so it binds to local port number that it will use during connection. This is usually assigned by system. If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote and endpoint set to the address and port of client. It needs a new socket for connection requests while trending to the needs of connected client. On the client side, if the connection is accepted a socket is successfully created and the client can use the socket to communicate with the server. The client and server can now communicate by writing to or reading from their sockets.

Thus flask-socketio is used to open sockets or event buckets for communication between client and server.

Message event handler on server-side (routes.py)

```
@socketio.on('message')
def handleMessage(msg):
    msgDict = json.loads(msg)
    msgToDeliver = {'sender':msgDict['sender'],
'receiver':msgDict['receiver'], 'content':msgDict['content'],
'timestamp':strftime("%I:%M %p", localtime()), 'room':msgDict['room']}
    print("\n\n{}\n\n".format(msgDict))
    if(msgDict['room'] == 'GLOBAL'):
        send(json.dumps(msgToDeliver), broadcast=True)
    elif msgDict['room'].lower() in ROOMS:
        send(json.dumps(msgToDeliver), room = msgToDeliver['room'])
    elif msgDict['room'].lower() not in ROOMS:      # One-to-one
        row = Rooms.query.filter_by(roomname=msgDict['room']).first()
        print("Message from {} to {}".format(msgDict['sender'],
msgDict['receiver']))
        row.count = row.count + 1
        print("row.count:", row.count)
        print("row.message", row.message)
        dict1 = json.loads(row.message)
        dict1[row.count] = msgDict
        row.message = json.dumps(dict1)
        db.session.commit()
        send(json.dumps(msgToDeliver), room = msgToDeliver['room'])
```

Load history event handler on client side (socketio.js)

```
socket.on('load_history', (msgHistory) => {
    msgHistory = JSON.parse(msgHistory);
    for(msgKey in msgHistory) {
        if(msgKey === '0') {
            continue;
        }
        else if(msgKey === '1') {
            if(roomName === msgHistory[msgKey]['room']) {

if(document.querySelector('#messages-list').childElementCount === 1) {
                console.log('Chatroom just opened, load history');
            }
        }
    }
}
```

```

    }
    else {
        console.log(`History already present, don't load
history`);
        return;
    }
    }
    else {
        console.log('History received was not of current room')
        break;
    }
}
let msgDisplay = document.createElement('h4');
msgDisplay.innerHTML = msgHistory[msgKey]['content'];
let timestamp = document.createElement('h6');
timestamp = msgHistory[msgKey]['timestamp'];
let listItem = document.createElement('div');
listItem.appendChild(msgDisplay);
document.getElementById('messages-list').appendChild(listItem);

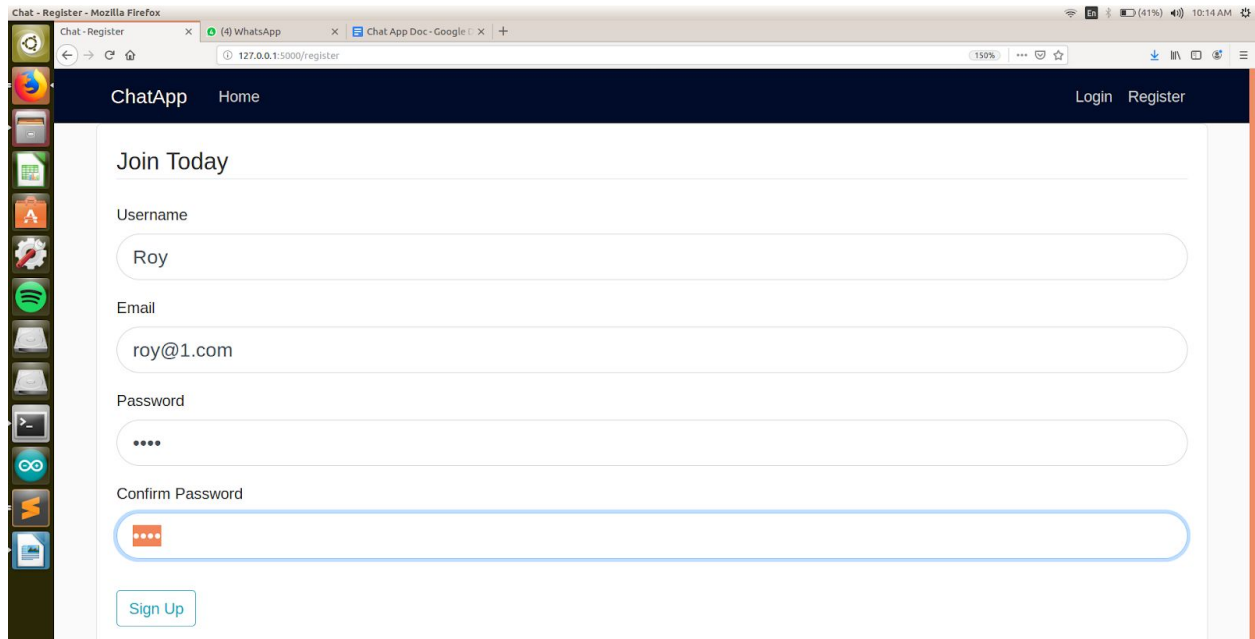
if(msgHistory[msgKey]['sender'] === username) {
    listItem.classList.add('meActivity');

    else {
        listItem.classList.add('themActivity');
    }
}

scrollDownChatWindow();
});

```

SCREENSHOTS



The screenshot shows the 'Chat - Register' page in a Mozilla Firefox browser. The address bar shows the URL '127.0.0.1:5000/register'. The page has a dark blue header with 'ChatApp' and 'Home' on the left, and 'Login' and 'Register' on the right. The main content area is titled 'Join Today' and contains a registration form with the following fields: 'Username' (containing 'Roy'), 'Email' (containing 'roy@1.com'), 'Password' (masked with dots), and 'Confirm Password' (also masked with dots). A 'Sign Up' button is located at the bottom of the form. The browser's taskbar on the left shows various application icons, and the system tray on the right shows the time as 10:14 AM.

Chat - Register - Mozilla Firefox

Chat - Register x (4) WhatsApp x Chat App Doc - Google x +

127.0.0.1:5000/register 150%

ChatApp Home Login Register

Join Today

Username

Roy

Email

roy@1.com

Password

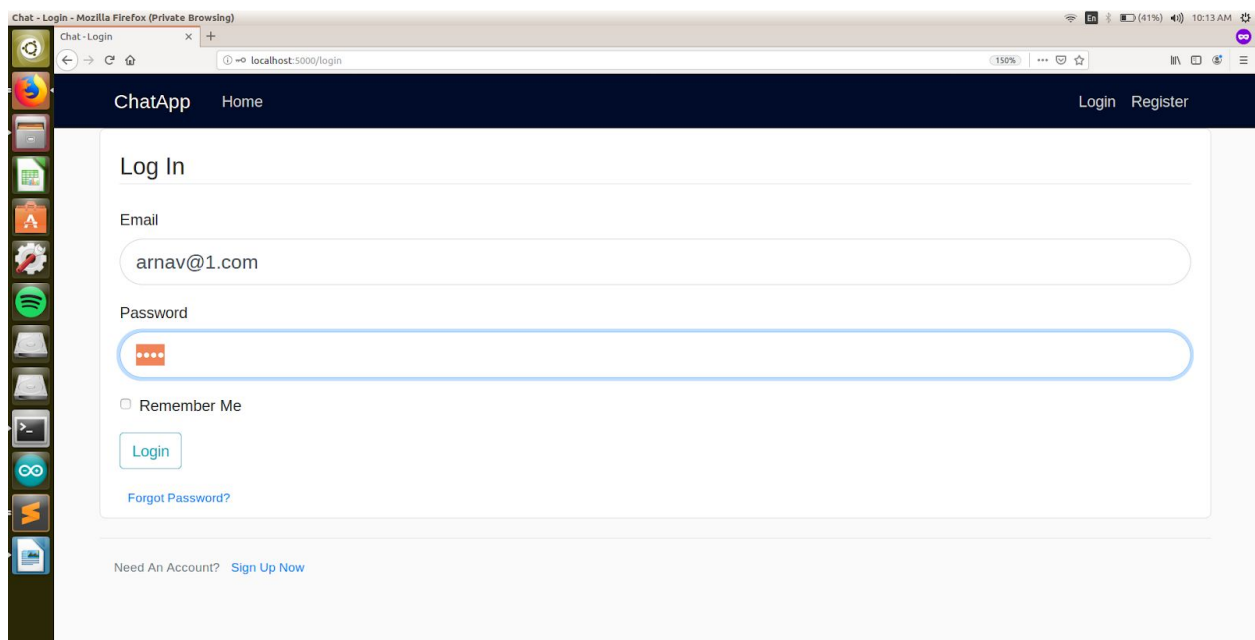
....

Confirm Password

....

Sign Up

Register screen



The screenshot shows the 'Chat - Login' page in a Mozilla Firefox browser. The address bar shows the URL 'localhost:5000/login'. The page has a dark blue header with 'ChatApp' and 'Home' on the left, and 'Login' and 'Register' on the right. The main content area is titled 'Log In' and contains a login form with the following fields: 'Email' (containing 'arnav@1.com') and 'Password' (masked with dots). Below the password field is a 'Remember Me' checkbox. A 'Login' button is located below the form. A link 'Forgot Password?' is also present. At the bottom of the page, there is a link 'Need An Account? Sign Up Now'. The browser's taskbar on the left shows various application icons, and the system tray on the right shows the time as 10:13 AM.

Chat - Login - Mozilla Firefox (Private Browsing)

Chat - Login x +

localhost:5000/login 150%

ChatApp Home Login Register

Log In

Email

arnav@1.com

Password

....

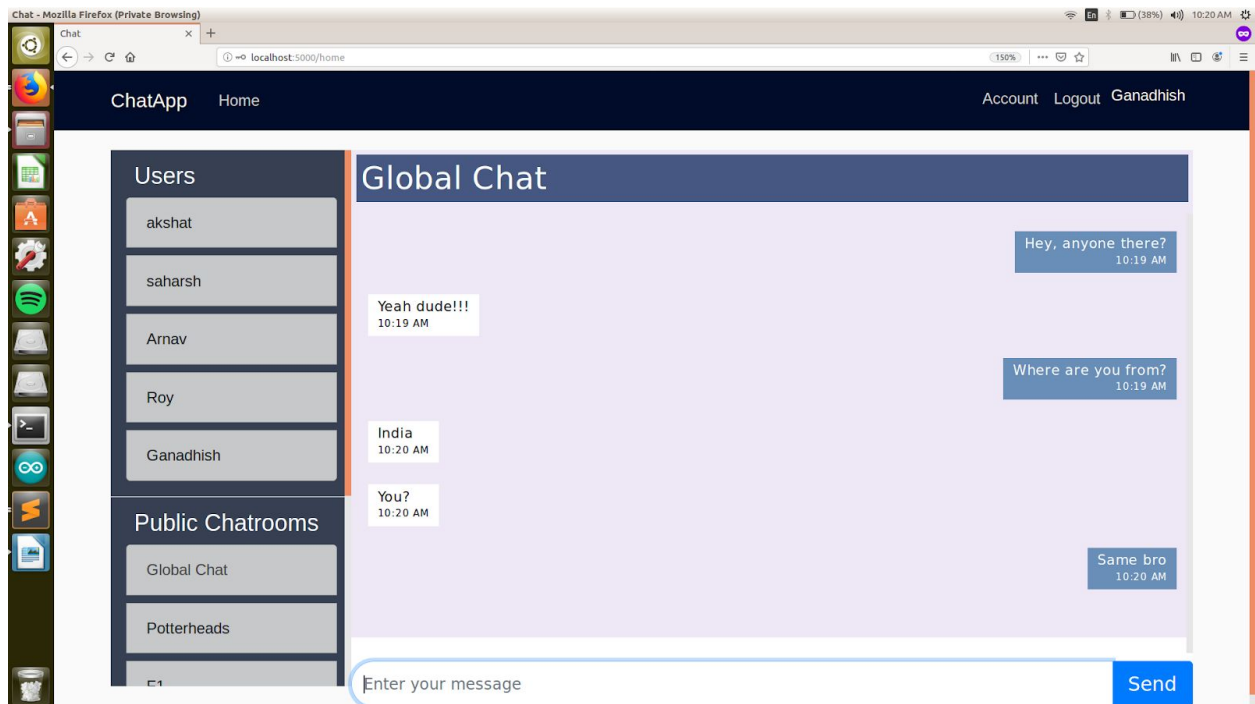
☐ Remember Me

Login

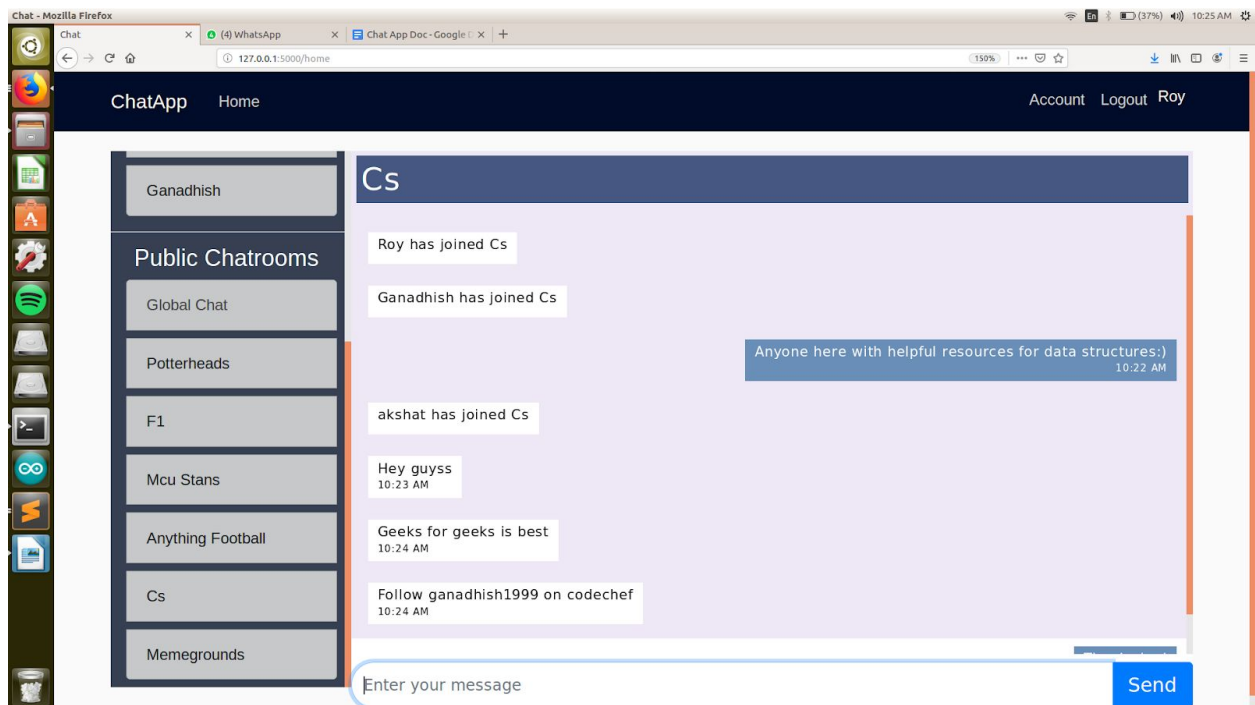
[Forgot Password?](#)

Need An Account? [Sign Up Now](#)

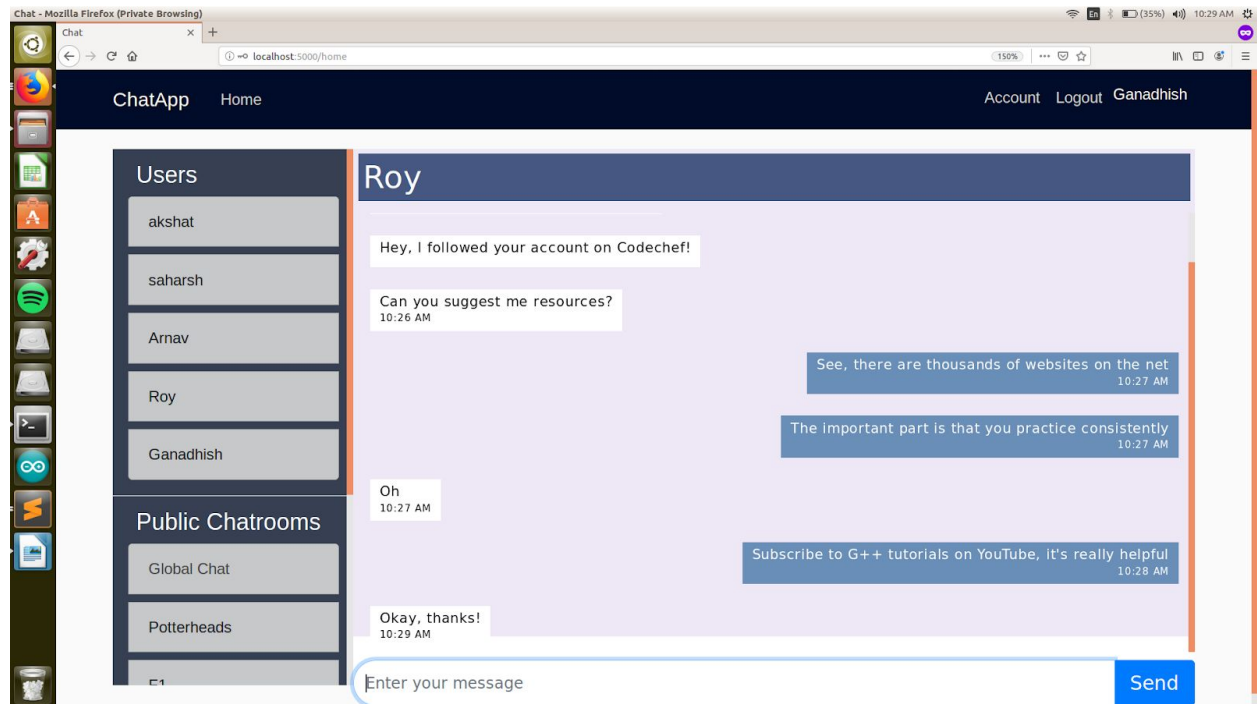
Login screen



Global chat



Public chatroom



Personal chat

FUTURE WORK

- Account profile page
- File sharing
- End to end encryption
- Password Strength Checker
- Chat Notifications and requests to connect
- Chatbot

GitHub Repository Link: <https://github.com/saharshleo/chattingApp/>