



VIT[®]

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

ONLINE PYTHON CHAT ROOM

A PROJECT REVIEW III REPORT

Submitted by

19BCE2609 Rishi Srikaanth

19BCE2607 Pritam Chaurasiya

CSE3001 Software Engineering

Guided by

Professor Bhavani S

Assistant Professor(Senior)

***School of Information Technology
and Engineering***

Table of Contents

S.No.	Title	Page No.
1	Detailed Problem Statement	1
2	Work Breakdown Structure	2
3	SRS	3-12
4	Gantt chart	13
5	Data flow diagram	14
6	ER - Diagram	15
7	UML Diagrams	16-23
	a. Use Case Diagram	16
	b. Class Diagram	17
	c. Sequence Diagram	18
	d. Collaboration Diagram	19
	e. Activity Diagram	20
	f. State Diagram	21
	g. Component Diagram	22
	h. Deployment Diagram	23
8	Test Cases	24-27
9	Algorithm & Flowchart	28-32
	a. Activity Chart for login	28
	b. Activity Chart for chat	29
	c. State Chart	30
	d. Context level Chart	31
	e. System Flow Chart	32
10	Complete Code	33-50
11	Results & Discussion	51
12	Complete Execution Screenshots	52-55

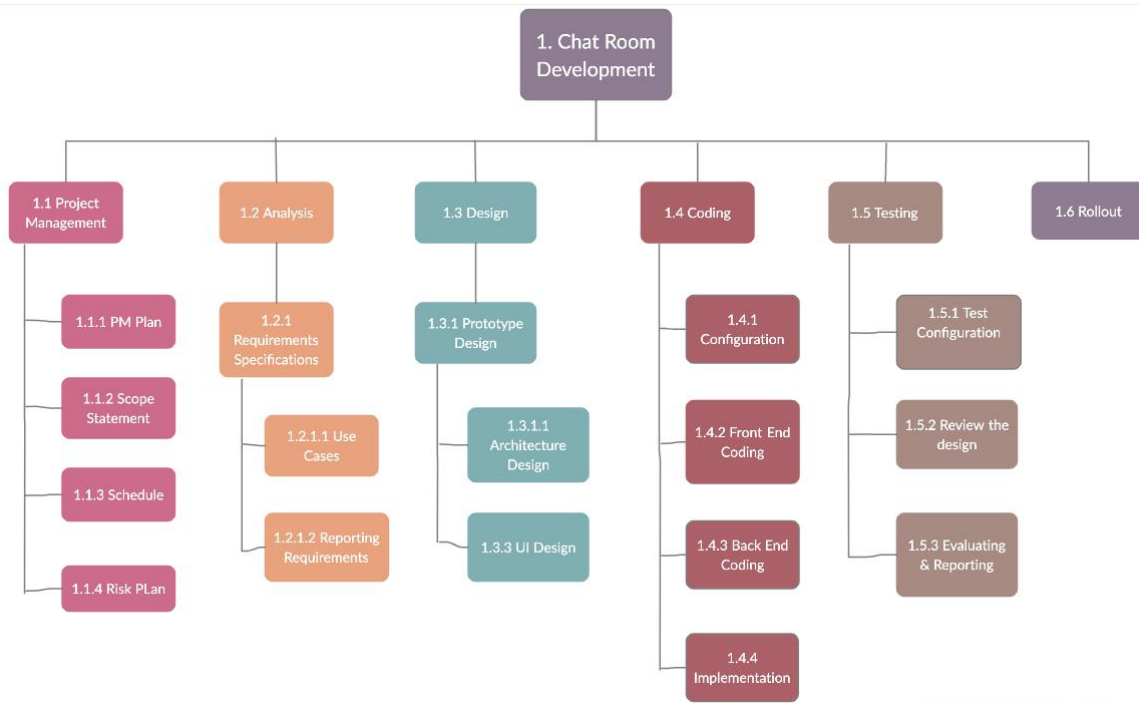
Problem Statement

Chat rooms have become a popular way to support a forum for n-way conversation or discussion among a set of people with interest in a common topic. Chat applications range from simple, text-based ones to entire virtual worlds with exotic graphics. In this project we are required to implement a simple text-based chat client/server application.

Email, newsgroup and messaging applications provide means for communication among people but these are one-way mechanisms and they do not provide an easy way to carry on a real-time conversation or discussion with people involved. Chat room extends the one-way messaging concept to accommodate multi-way communication among a set of people.

This project is to create a chat application with a server and clients to enable the clients to chat with many other clients in the same common chat group. This project is to simulate the multicast chatting. In the case of multicasting when a message is sent to a group of clients, then only a single message is sent to the router. Which means that the client will send the message only once and based on the location of the clients, the router will either pass the message to another router or if the clients are in the same local network the router will send a copy of the message to each client in that network. So this way this becomes a light weight chat application with less number of messages being passed around in the network that can be used in the computationally demanding scenarios such as gaming.

WORK BREAKDOWN STRUCTURE



Software Requirements Specification

for

Light Weight Python Chatroom For Gaming

Version 1.0

Prepared by

Group Name: *<place your group name here>*

Rishi Srikaanth	19BCE2609	rishi.srikaanth2019@vitstudent.ac.in
Pritam Chaurasia	19BCE2607	pritam.chaurasiya2019@vitstudent.ac.in
<name>	<student #>	<e-mail>
<name>	<student #>	<e-mail>
<name>	<student #>	<e-mail>

Instructor: Prof. Bhavani Sundar Raj

Course: CSE3001 : Software Engineering

Lab Section: L37+L38

Teaching Assistant: NA

Date: 5/11/2020

Contents

CONTENTS	II
REVISIONS	II
1. INTRODUCTION	1
1.1. DOCUMENT PURPOSE	1
1.2. PRODUCT SCOPE	1
1.3. INTENDED AUDIENCE AND DOCUMENT OVERVIEW	1
1.4. DEFINITIONS, ACRONYMS AND ABBREVIATIONS	1
1.5. DOCUMENT CONVENTIONS	2
1.6. REFERENCES AND ACKNOWLEDGMENTS	2
2. OVERALL DESCRIPTION	3
2.1. PRODUCT FUNCTIONALITY	3
2.2. DESIGN AND IMPLEMENTATION CONSTRAINTS	3
2.3. ASSUMPTIONS AND DEPENDENCIES	4
3. SPECIFIC REQUIREMENTS	5
3.1. EXTERNAL INTERFACE REQUIREMENTS	5
3.2. FUNCTIONAL REQUIREMENTS	6
3.3. USE CASE MODEL	7
4. OTHER NON-FUNCTIONAL REQUIREMENTS	8
4.1. PERFORMANCE REQUIREMENTS	8
4.2. SAFETY AND SECURITY REQUIREMENTS	8
4.3. SOFTWARE QUALITY ATTRIBUTES	8

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
2.2	Product File Name	Information about the revision. This table does not have to be filled in whenever a document is touched, only when the version is being upgraded.	00/00/00
2.3	Design and Implementation Constraints		
2.4	Assumptions and Dependencies		

1. Introduction

Light Weight Python Chatroom For Gaming is an easy-to-use, server based LAN messaging application for effective communication. It correctly identifies and works under all operating systems with unlimited user accounts and is the only secure, messenger.

1.1. Document Purpose

This document is to explain and describe the workings of the product : Light weight Python Chatroom.

1.2. Product Scope

Mail Server with Live Chat is going to be a text communication software, it will be able to communicate between two computers using point to point communication. .

The limitation of Live Chat is it does not support audio conversations. To overcome this limitation we are concurrently working on developing better technologies.

Data communication over LAN networks is going to be a most important mode of communication for the corporates in the very near future. Companies have to rely on external networks which not only are unreliable but also cost inefficient.

Companies would like to have a communication software wherein they can communicate instantly within their organization.

The fact that the software uses an internal network setup within the organization makes it very secure from outside attacks.

1.3. Intended Audience and Document Overview

Professor overseeing this project.

1.4. Definitions, Acronyms and Abbreviations

NA

1.5. Document Conventions

IEEE : This style manual provides editorial guidelines for *IEEE Transactions, Journals, and Letters*. For spelling reference, *IEEE Publications* uses *Webster's College Dictionary, 4th Edition*. For guidance on grammar and usage not included in this manual, please consult *The Chicago Manual of Style*, published by the University of Chicago Press.

1.6. References and Acknowledgments

Final Project Report

2. Overall Description

2.1. Product Overview

Chat rooms have become a popular way to support a forum for n-way conversation or discussion among a set of people with interest in a common topic. Chat applications range from simple, text-based ones to entire virtual worlds with exotic graphics. In this project we are required to implement a simple text-based chat client/server application.

Email, newsgroup and messaging applications provide means for communication among people but these are one-way mechanisms and they do not provide an easy way to carry on a real-time conversation or discussion with people involved. Chat room extends the one-way messaging concept to accommodate multi-way communication among a set of people.

This project is to create a chat application with a server and clients to enable the clients to chat with many other clients in the same common chat group. This project is to simulate the multicast chatting. In the case of multicasting when a message is sent to a group of clients, then only a single message is sent to the router. Which means that the client will send the message only once and based on the location of the clients, the router will either pass the message to another router or if the clients are in the same local network the router will send a copy of the message to each client in that network. So this way this becomes a light weight chat application with less number of messages being passed around in the network that can be used in the computationally demanding scenarios such as gaming.

2.2. Product Functionality-

- Communication: To develop an instant messaging solution to enable users to seamlessly communicate with each other.
- User friendliness: The project should be very easy to use enabling even a novice person to use it.

2.3. Design and Implementation Constraints

Implementation is the stage of the project where the theoretical design is turned in to a working system. The implementation state is a system project in its own right.

It involves careful planning, investigation of the current system and its constraints on implementation, design of methods to achieve the changeover, training of staff in the change over procedure and evaluation of change over methods.

Once the planning has been completed, the major efforts are to ensure that the program in the system is working properly.

At the same time concentrate on training user staff. When the staff has been trained a full system can carry out.

2.4. Assumptions and Dependencies

- There should be LAN or internet connection.
- Clients should know each other.
- There can be multiple clients.

3. Specific Requirements

3.1. External Interface Requirements

3.1.1. User Interfaces

This application interacts with the user through G.U.I. The interface is simple , easy to handle and self-explanatory.

- Once opened, user will easily come into the flow with the application and easily uses all interfaces properly.
- However the basic interface available in our application is
- Title panel
- Content panel
- Admin panel.

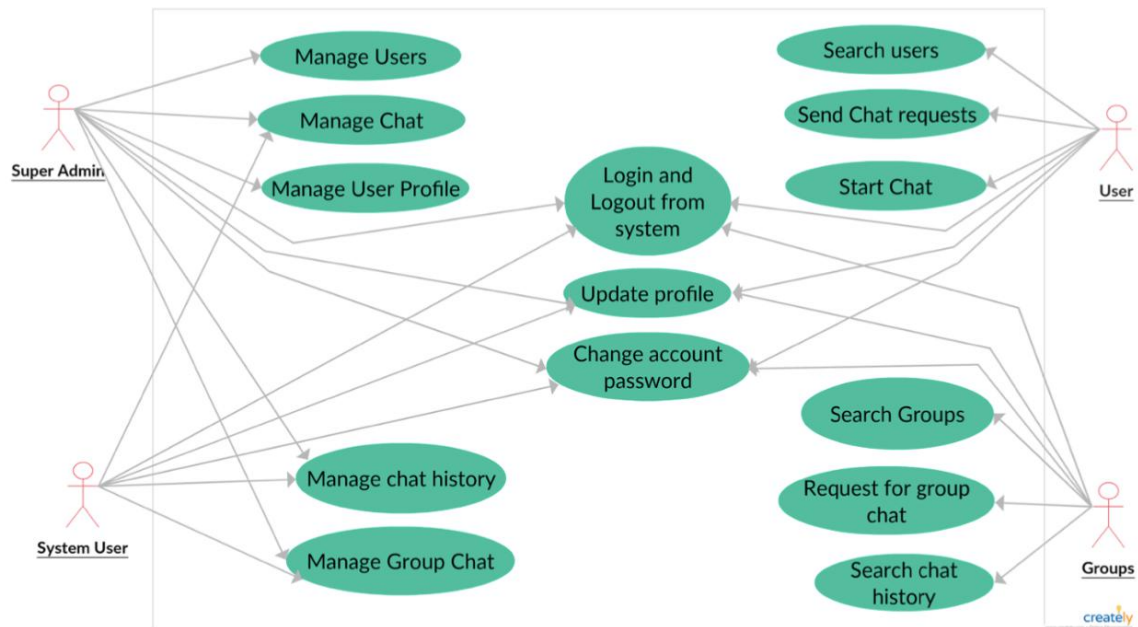
3.1.2. Hardware Interfaces

- 128 MB RAM required.
- Processor with speed of 500 MHz.
- Internet or LAN connection.
- MOUSE: 2 or 3 button mouse
- KEYBOARD: 101 key Keyboard

3.1.3. Software Interfaces

- Notepad++ is a text editor and source code editor and provides an environment for developing HTML, jsp, JavaScript many other editing purposes.
- Coding done in java so required JDK 1.4 and above for run java programs.
- Operating system (such as window 7, 8, xp, Linux etc).

3.2. Functional Requirements



3.2.1. F1: The system shall ...

- There is a two way communication between different clients and server.
- This chat application can be used for group discussion.
- It allows users to find other logged in users.

3.3. Use Case Model

3.3.1. Use Case #1 Game Chat

Author – Rishi Srikaanth

Purpose - Provides a light weight lan based chatroom application that can be used as a chatting application while gaming

Requirements Traceability –Minimum software and hardware requirements stated above

Priority - High

Preconditions - NA

Post conditions -NA

Actors – Host client (Server), user clients (clients)

Extends – NA

Flow of Events

1. Basic Flow - Host fires up the server - clients join the server by logging in - they start playing
2. Alternative Flow - Host starts the server - it fails to start - Another client acts as host - rest of the clients join the server - they start playing
3. Exceptions - No exceptions other than system bottle necks

4. Other Non-functional Requirements

4.1. Performance Requirements

NA

4.2. Safety and Security Requirements

NA

4.3. Software Quality Attributes

Correctness

These requirements deal with the correctness of the output of the software system. They include –

- Output mission
- The required accuracy of output that can be negatively affected by inaccurate data or inaccurate calculations.
- The completeness of the output information, which can be affected by incomplete data.
- The up-to-dateness of the information defined as the time between the event and the response by the software system.
- The availability of the information.
- The standards for coding and documenting the software system.

Reliability

Reliability requirements deal with service failure. They determine the maximum allowed failure rate of the software system, and can refer to the entire system or to one or more of its separate functions.

Efficiency

It deals with the hardware resources needed to perform the different functions of the software system. It includes processing capabilities (given in MHz), its storage capacity (given in MB or GB) and the data communication capability (given in MBPS or GBPS).

It also deals with the time between recharging of the system's portable units, such as, information system units located in portable computers, or meteorological units placed outdoors.

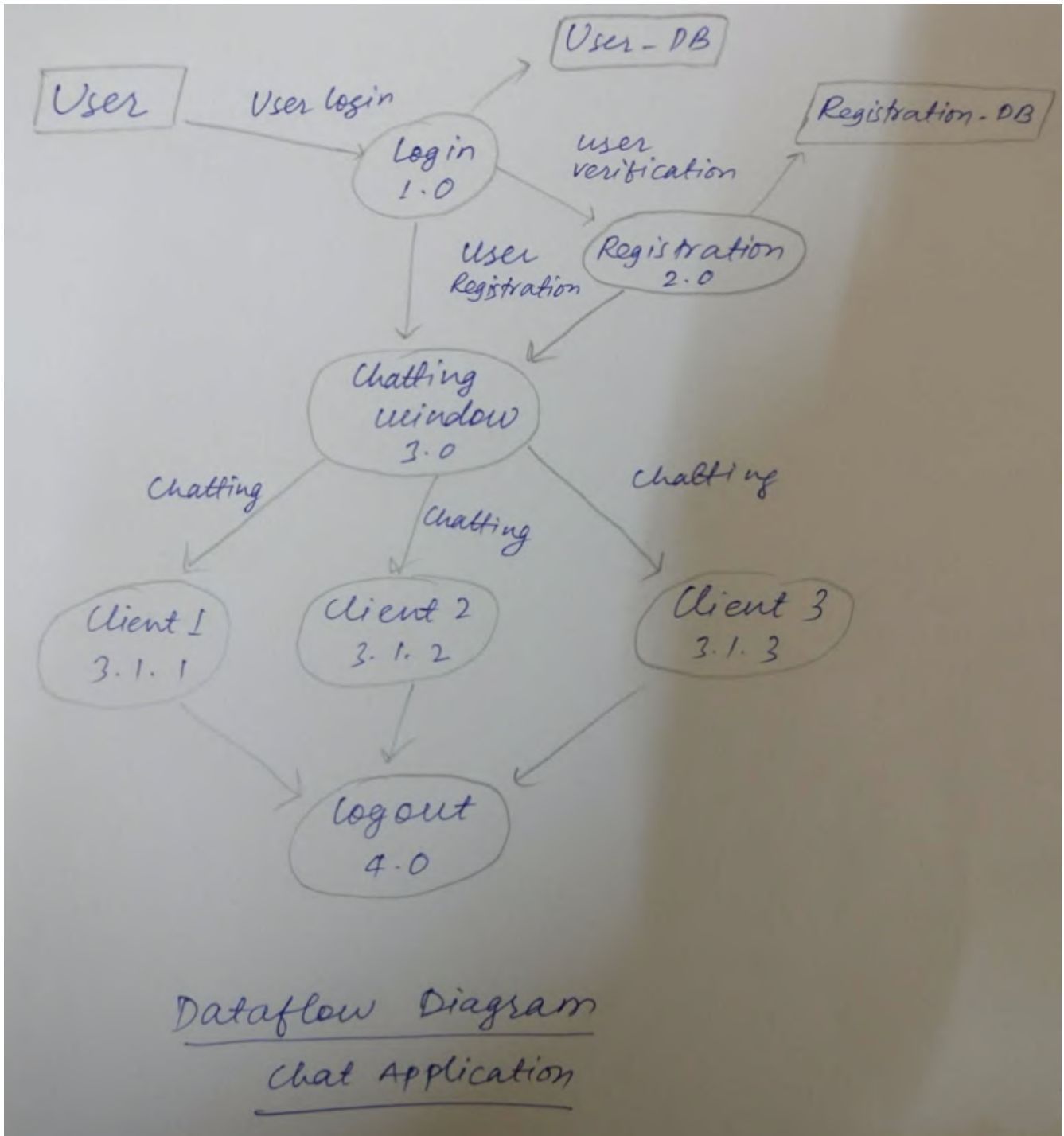
Integrity

This factor deals with the software system security, that is, to prevent access to unauthorized persons, also to distinguish between the group of people to be given read as well as write permit.

Usability

Usability requirements deal with the staff resources needed to train a new employee and to operate the software system.

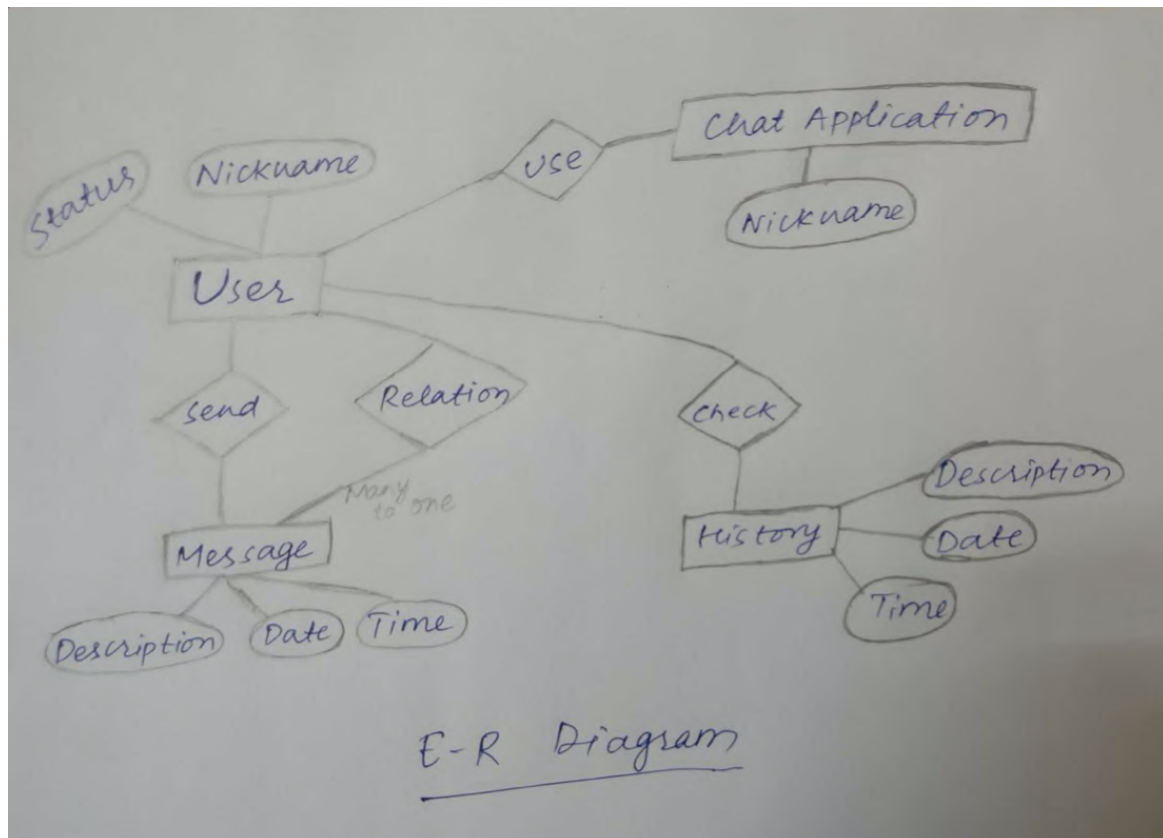
Data flow diagram (DFD)



GANTT Chart

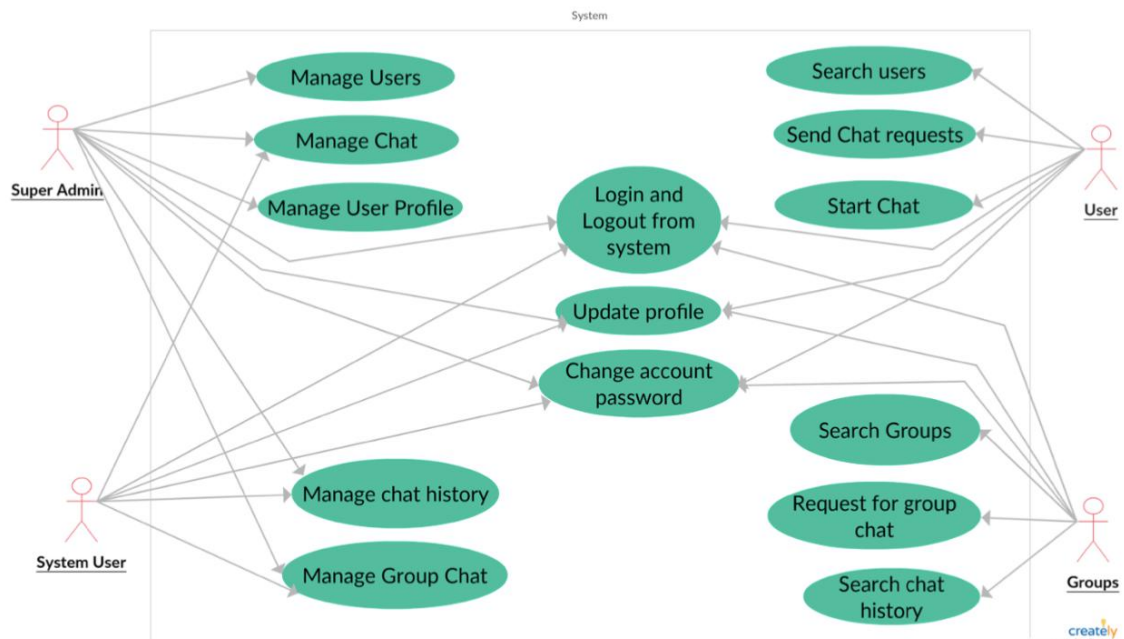
[illegible]

ER diagram

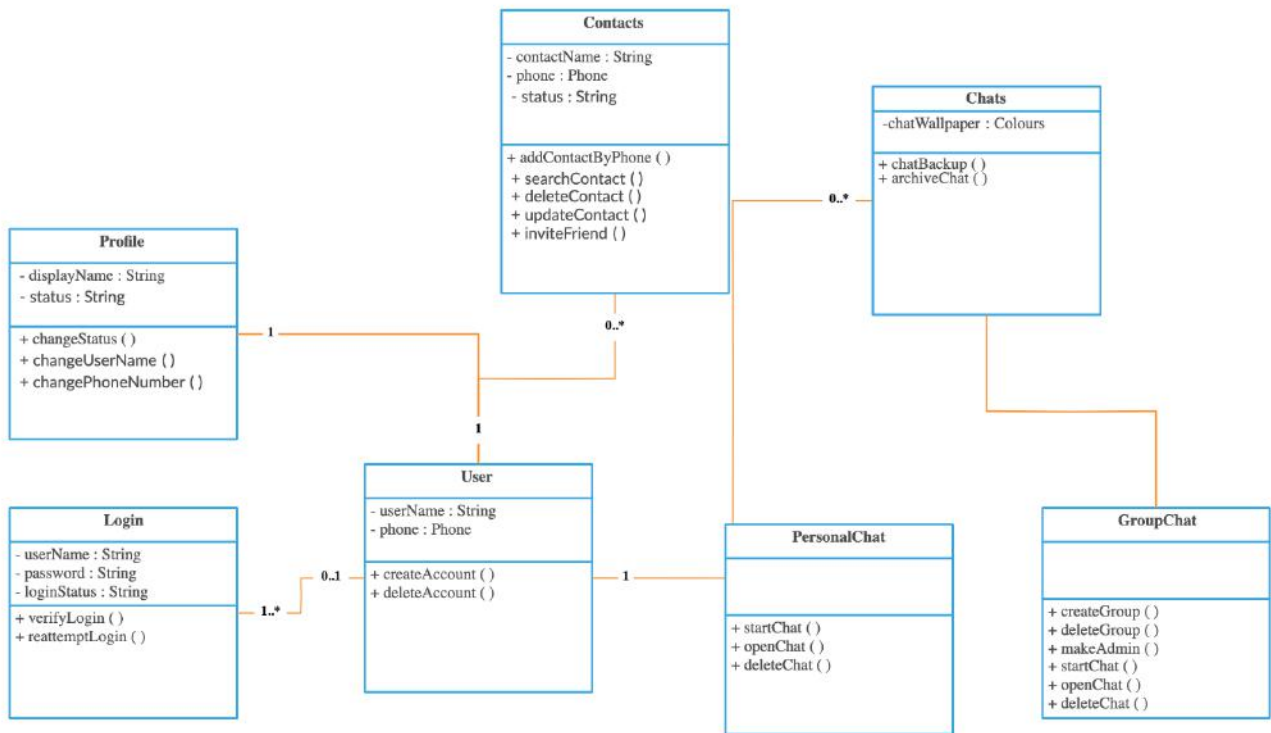


UML Diagrams

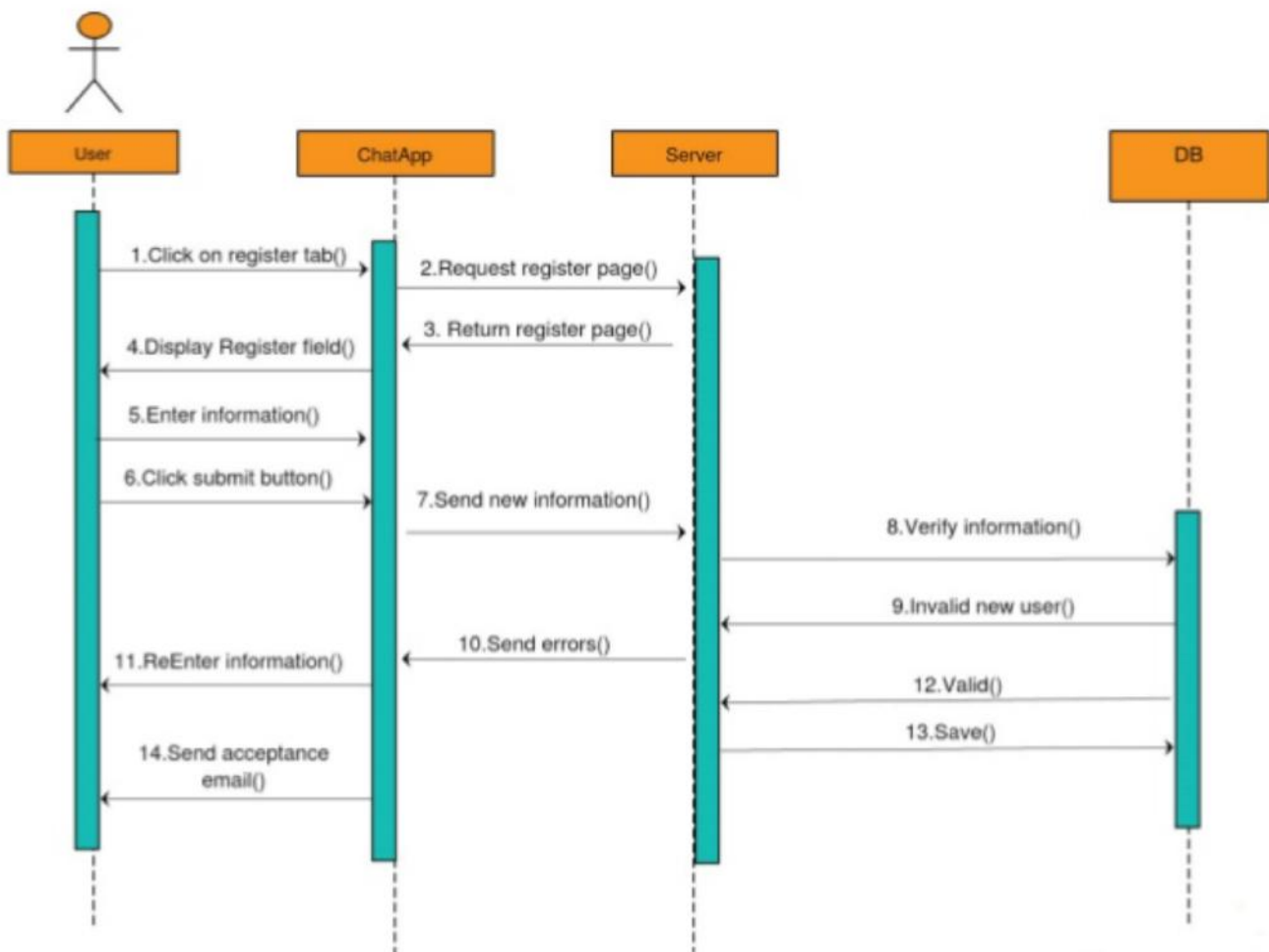
1. Use Case Diagram



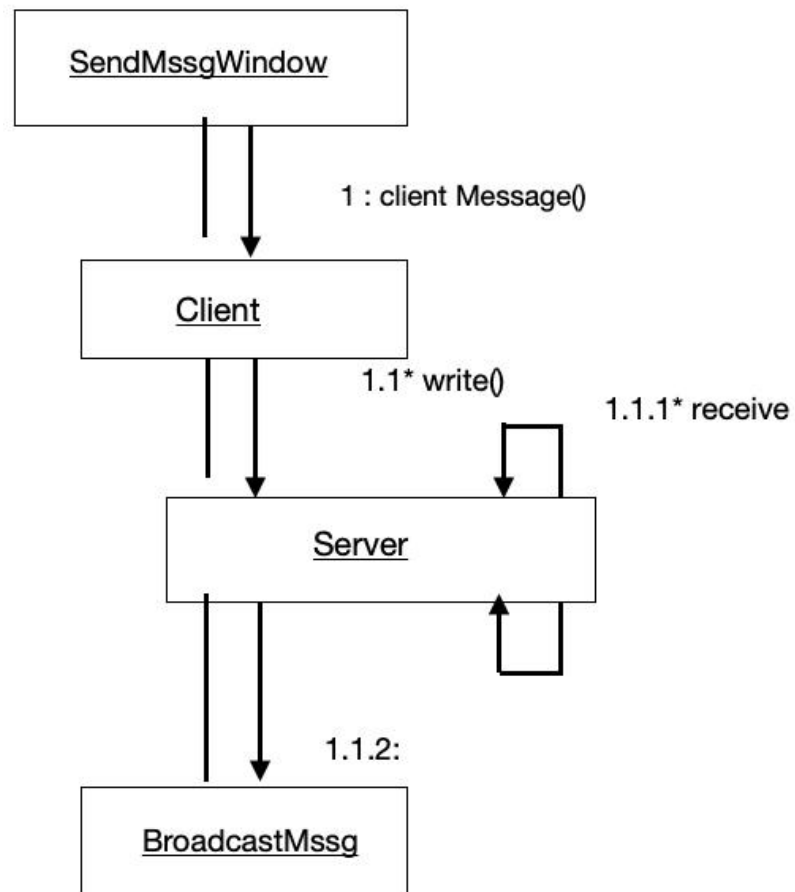
2. Class Diagram



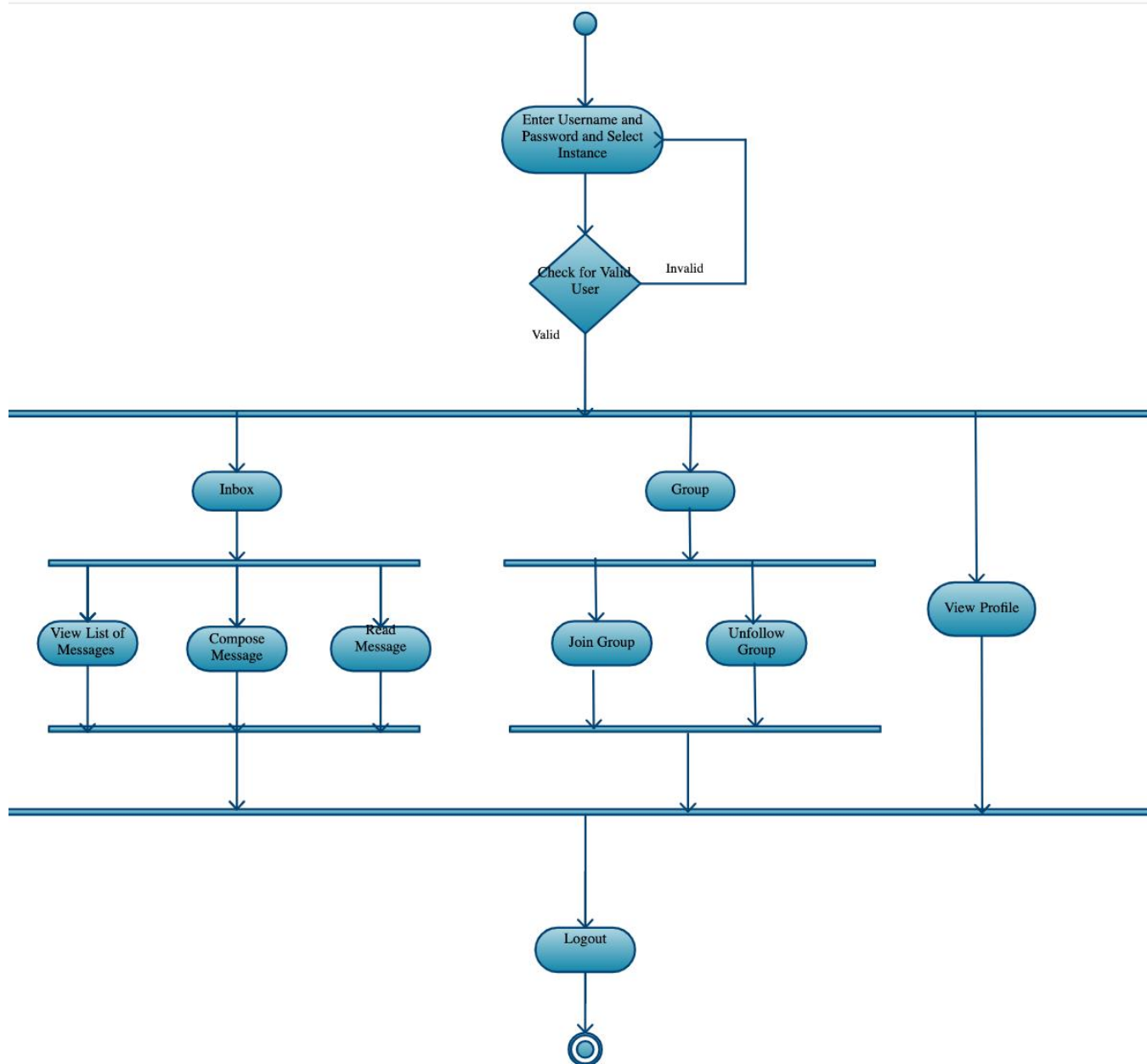
3. Sequence Diagram



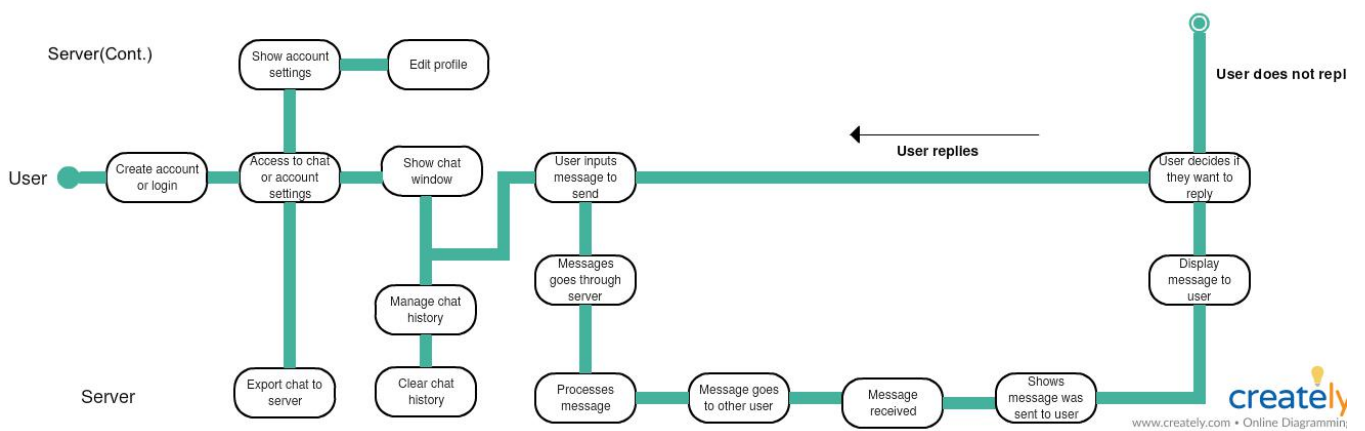
4. Collaboration Diagram



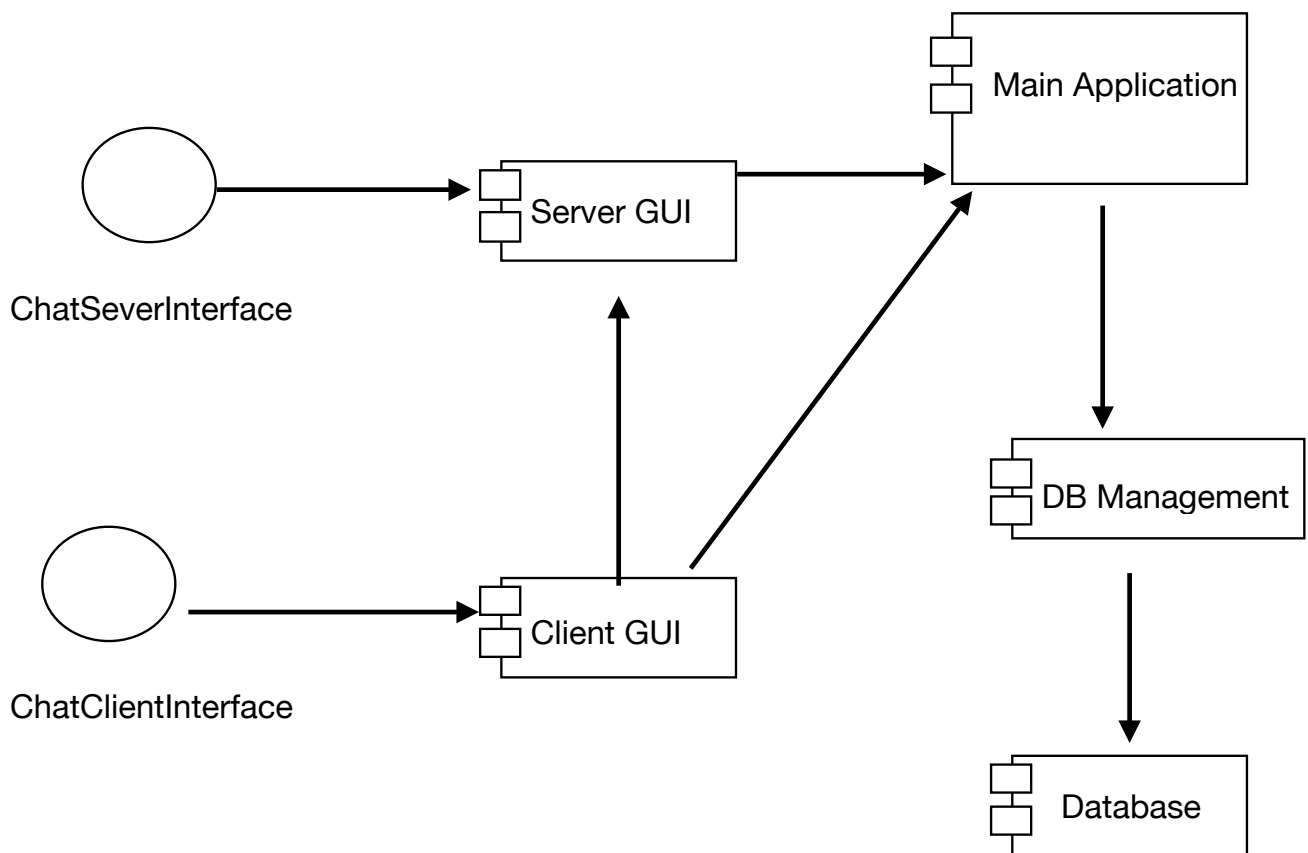
5. Activity Diagram



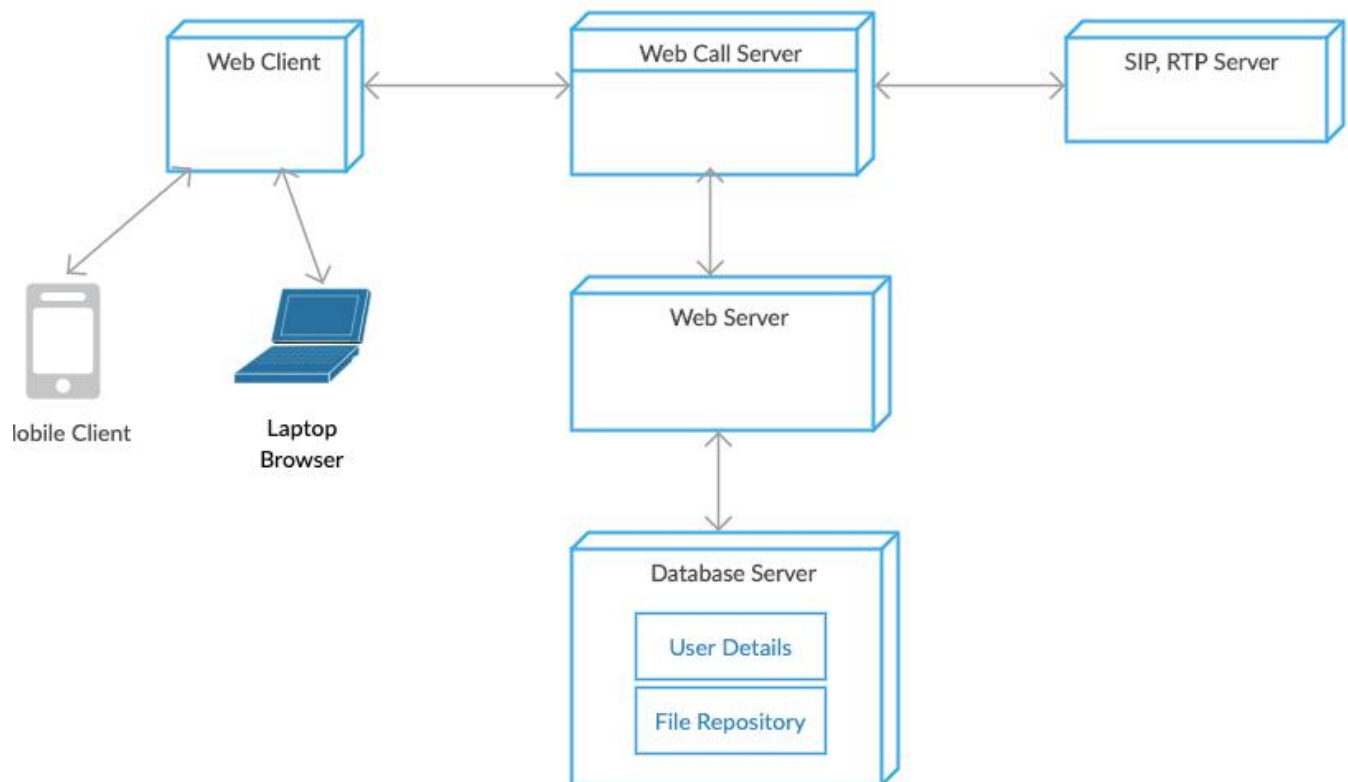
6. State Diagram



7. Component Diagram



8. Deployment Diagram



Test Case For Chat Application

Test Scenario ID		Login 1		Test Case ID		Login - 1A	
Test Case Description		Login - Positive case		Test Priority		High	
Pre-Requisite		NA		Post-Requisite		NA	
Test Execution Steps:							
S.No	Action	Inputs	Expected Output	Actual Output	Test Browser	Test Result	
1	Launch Application	Enter Website URL	Chatroom homepage	Chatroom homepage	Chrome	Pass	
2	Enter username & password	Username : user1 Password : *****	Login success	Login success	Chrome	Pass	

Test Scenario ID	Login 1	Test Case ID	Login - 1B			
Test Case Description	Login - Negative case	Test Priority	High			
Pre-Requisite	NA	Post-Requisite	NA			
Test Execution Steps:						
S.No	Action	Inputs	Expected Output	Actual Output	Test Browser	Test Result
1	Launch Application	Enter Website URL	Chatroom homepage	Chatroom homepage	Chrome	Pass

2	Enter invalid username & any Password	Username : user1 Password : *****	The email username that you've entered doesn't match any account. Sign up for an account.	The email username that you've entered doesn't match any account. Sign up for an account.	Chrome	Pass
3	Enter valid username & incorrect Password	Username : user1 Password : *****	The password that you've entered is incorrect. Forgotten password?	The password that you've entered is incorrect. Forgotten password?	Chrome	Pass

Test Scenario ID	Type your nickname 2	Test Case ID	Type your nickname 2A			
Test Case Description	Type your nickname - Positive case	Test Priority	High			
Pre-Requisite	Login - Positive case	Post-Requisite	NA			
Test Execution Steps:						
S.No	Action	Inputs	Expected Output	Actual Output	Test Browser	Test Result
1	Enter valid nickname	Nickname : nickname1	Enters the chat	Enters the chat	Chrome	Pass

Test Scenario ID	Type your nickname 2	Test Case ID	Type your nickname 2B			
Test Case Description	Type your nickname - Negative case	Test Priority	High			
Pre-Requisite	Login - Positive case	Post-Requisite	NA			
Test Execution Steps:						
S.No	Action	Inputs	Expected Output	Actual Output	Test Browser	Test Result
1	Proceeding without entering nickname	Null	Please enter a nickname to proceed	Please enter a nickname to proceed	Chrome	Pass

Test Scenario ID	Send Message 3	Test Case ID	Send Message 3A			
Test Case Description	Type your nickname - Positive case	Test Priority	High			
Pre-Requisite	Type your nickname - Positive case	Post-Requisite	NA			
Test Execution Steps:						
S.No	Action	Inputs	Expected Output	Actual Output	Test Browser	Test Result
1	Enter a simple message	Hello	The message "Hello" is sent.	The message "Hello" is sent	Chrome	Pass

Test Scenario ID	Send Message 3	Test Case ID	Send Message 3B			
Test Case Description	Type your nickname - Negative case	Test Priority	High			
Pre-Requisite	Type your nickname - Positive case	Post-Requisite	NA			
Test Execution Steps:						
S.No	Action	Inputs	Expected Output	Actual Output	Test Browser	Test Result
1	Clicking on send button without entering a message	NA	It doesn't send anything.	It doesn't send anything.	Chrome	Pass

Test Scenario ID	Send Message 3	Test Case ID	Send Message 3C			
Test Case Description	Type your nickname - Negative case	Test Priority	High			
Pre-Requisite	Type your nickname - Positive case	Post-Requisite	NA			
Test Execution Steps:						
S.No	Action	Inputs	Expected Output	Actual Output	Test Browser	Test Result
1	Clicking on send button without entering a message	NA	It doesn't send anything.	It doesn't send anything.	Chrome	Pass

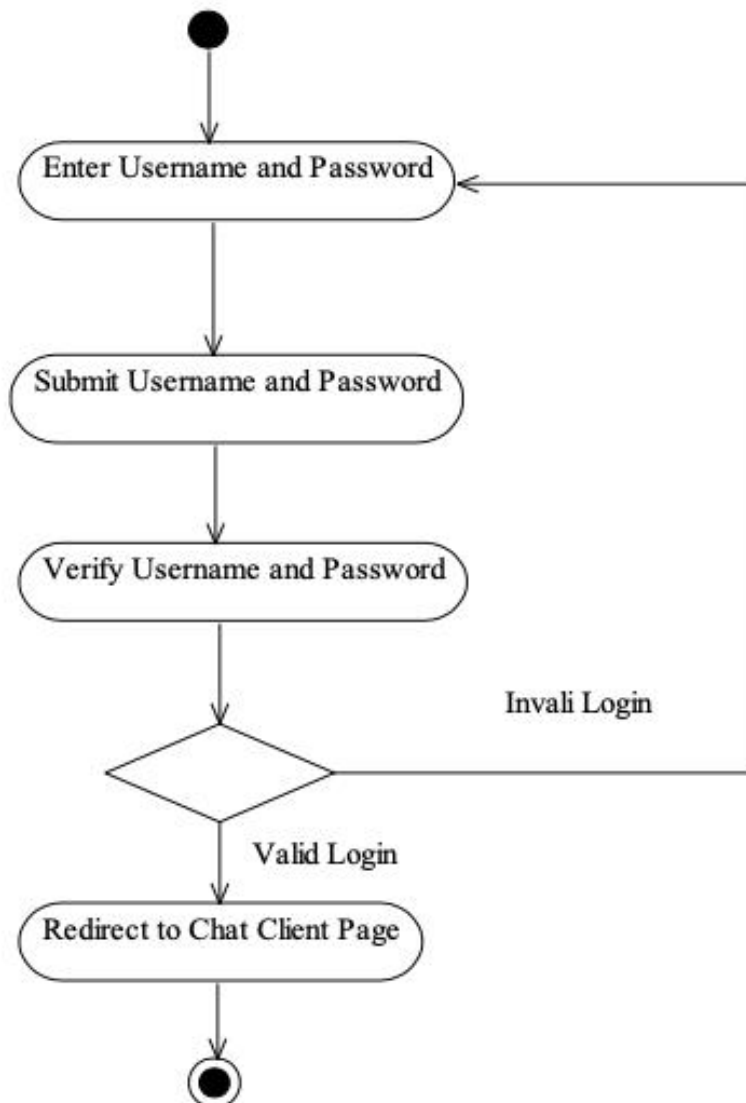
Test Scenario ID	Send Message 3	Test Case ID	Send Message 3D			
Test Case Description	Type your nickname - Positive case	Test Priority	High			
Pre-Requisite	Type your nickname - Positive case	Post-Requisite	NA			
Test Execution Steps:						
S.No	Action	Inputs	Expected Output	Actual Output	Test Browser	Test Result
1	Typing a message of 250 characters.	A message of 250 characters	The message is sent.	The message is sent.	Chrome	Pass

Test Scenario ID	Send Message 3	Test Case ID	Send Message 3E			
Test Case Description	Type your nickname - Positive case	Test Priority	High			
Pre-Requisite	Type your nickname - Negative case	Post-Requisite	NA			
Test Execution Steps:						
S.No	Action	Inputs	Expected Output	Actual Output	Test Browser	Test Result
1	Typing a message with more than 250 characters.	A message with more than 250 characters	The message is not sent.	The message is not sent.	Chrome	Pass

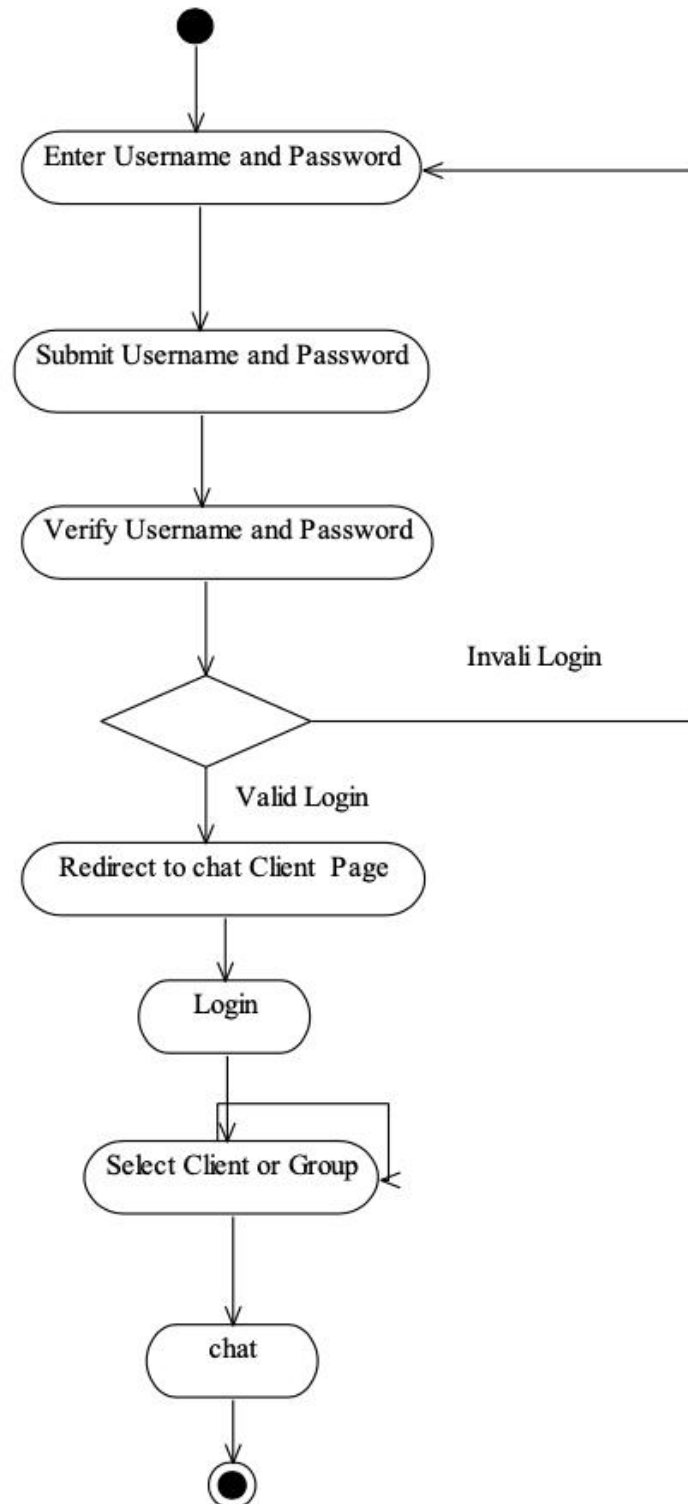
Test Scenario ID	Log out 4	Test Case ID	Send Message 4A			
Test Case Description	Log out - Positive case	Test Priority	High			
Pre-Requisite	Log in - Positive case	Post-Requisite	NA			
Test Execution Steps:						
S.No	Action	Inputs	Expected Output	Actual Output	Test Browser	Test Result
1	Clicking on log out button.	NA	You return to login page.	You return to login page.	Chrome	Pass

Algorithms & Flow Charts

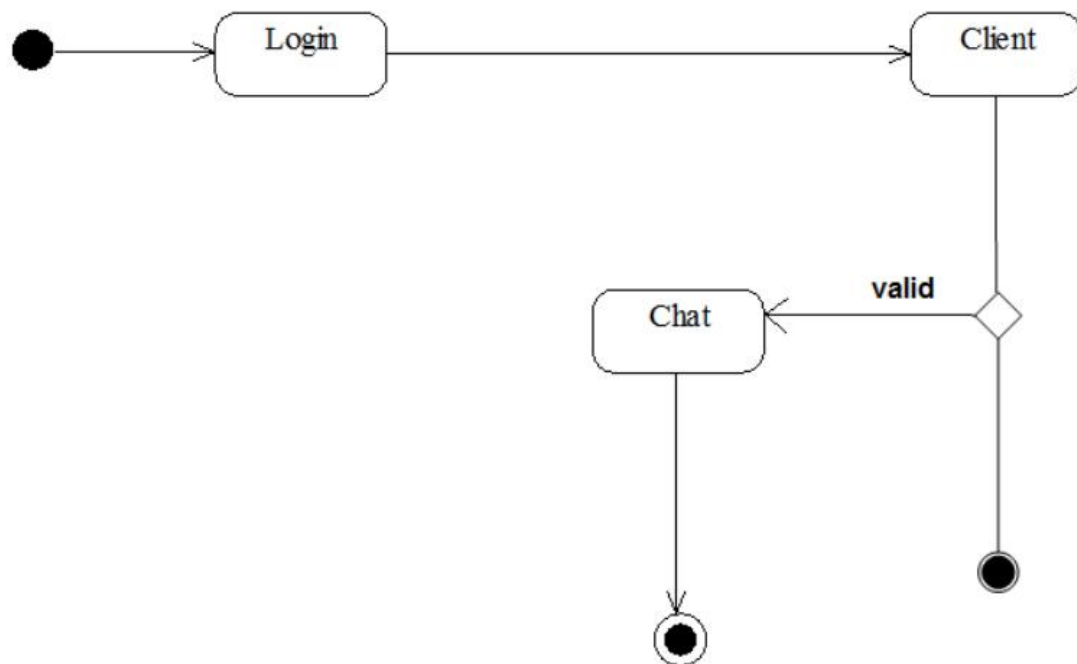
Activity Diagram for login



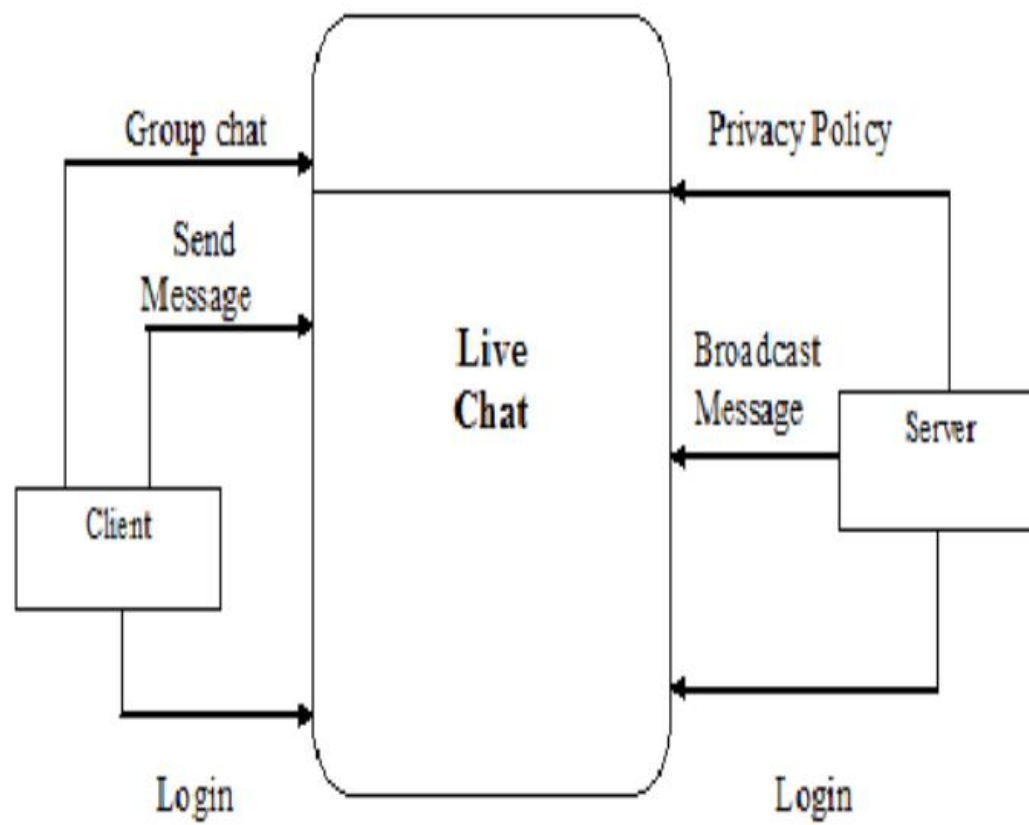
Activity Diagram for Chat



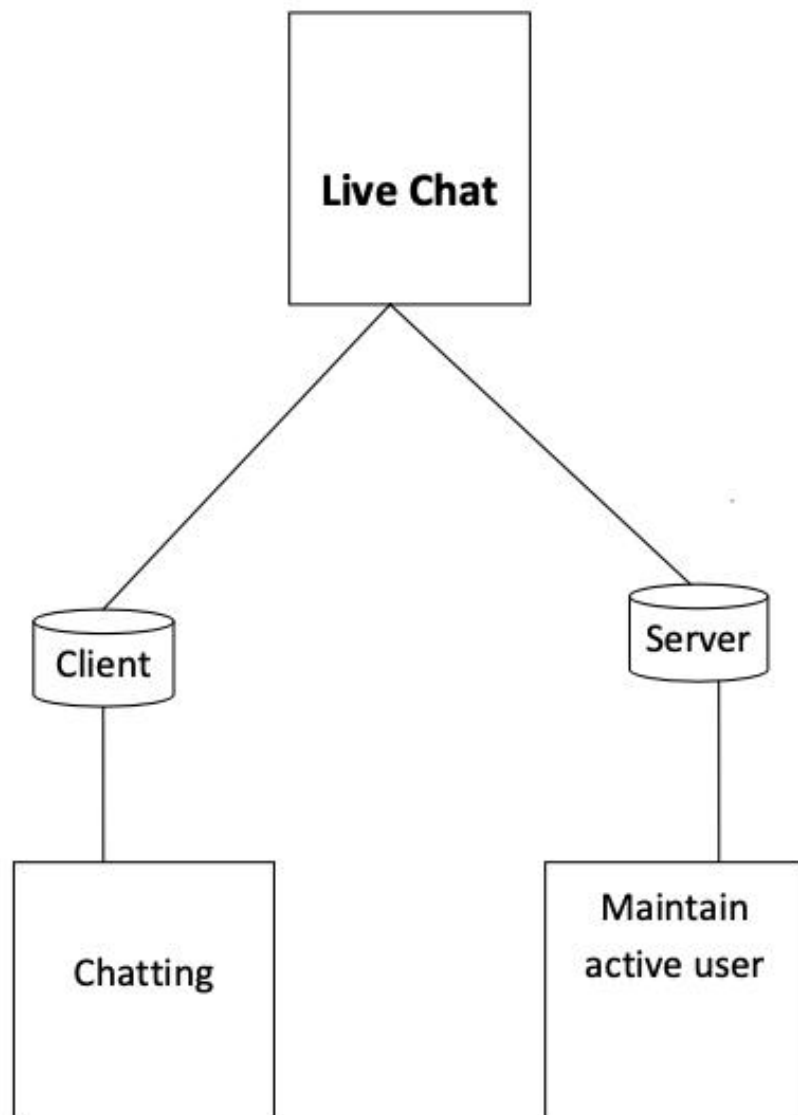
State chart diagram



Context Level Chart



System FlowChart



Complete Code

Github

https://github.com/RoyaleRishi/Python_Chatroom

Screenshots

config.py : SE Project/Chatroom

```
from dotenv import load_dotenv
from pathlib import Path # python3 only
import os

# set path to env file
env_path = Path('.') / '.env'
load_dotenv(dotenv_path=env_path)

class Config:
    """Set Flask configuration vars from .env file."""

    # Load in environment variables
    TESTING = os.getenv('TESTING')
    FLASK_DEBUG = os.getenv('FLASK_DEBUG')
    SECRET_KEY = os.getenv('SECRET_KEY')
    SERVER = os.getenv('SERVER')
```

main.py : SE Project/Chatroom

```
from flask import session
from flask_socketio import SocketIO
import time
from application import create_app
from application.database import DataBase
import config

# SETUP
app = create_app()
socketio = SocketIO(app) # used for user communication

# COMMUNICATION FUNCTIONS

@socketio.on('event')
def handle_my_custom_event(json, methods=['GET', 'POST']):
    """
    handles saving messages once received from web server
    and sending message to other clients
    :param json: json
    :param methods: POST GET
    :return: None
    """
    data = dict(json)
    if "name" in data:
        db = DataBase()
        db.save_message(data["name"], data["message"])

    socketio.emit('message response', json)

# MAINLINE

if __name__ == "__main__": # start the web server
    socketio.run(app, debug=True, host=str(config.Config.SERVER))
```

`__init__.py` : SE Project/Chatroom/application

```
from flask import Flask

def create_app():
    """Construct the core application."""
    app = Flask(__name__, instance_relative_config=False)
    app.config.from_object('config.Config')

    with app.app_context():
        # Imports
        from .views import view
        from .filters import _slice
        from .database import DataBase

        # REGISTER ROUTES
        app.register_blueprint(view, url_prefix="/")

        # REGISTER CONTEXT PROCESSOR
        @app.context_processor
        def slice():
            return dict(slice=_slice)

    return app
```

database.py : SE Project/Chatroom/application. (1)

```
import sqlite3
from sqlite3 import Error
from datetime import datetime
import time

# CONSTANTS

FILE = "messages.db"
PLAYLIST_TABLE = "Messages"

class DataBase:
    """
    used to connect, write to and read from a local sqlite3 database
    """
    def __init__(self):
        """
        try to connect to file and create cursor
        """
        self.conn = None
        try:
            self.conn = sqlite3.connect(FILE)
        except Error as e:
            print(e)

        self.cursor = self.conn.cursor()
        self._create_table()

    def close(self):
        """
        close the db connection
        :return: None
        """
        self.conn.close()

    def _create_table(self):
        """
        create new database table if one doesn't exist
        :return: None
        """
        query = f"""CREATE TABLE IF NOT EXISTS {PLAYLIST_TABLE}
        (name TEXT, content TEXT, time Date, id INTEGER PRIMARY KEY AUTOINCREMENT)"""
        self.cursor.execute(query)
        self.conn.commit()
```

database.py : SE Project/Chatroom/application. (2)

```
def get_all_messages(self, limit=100, name=None):
    """
    returns all messages
    :param limit: int
    :return: list[dict]
    """
    if not name:
        query = f"SELECT * FROM {PLAYLIST_TABLE}"
        self.cursor.execute(query)
    else:
        query = f"SELECT * FROM {PLAYLIST_TABLE} WHERE NAME = ?"
        self.cursor.execute(query, (name,))

    result = self.cursor.fetchall()

    # return messages in sorted order by date
    results = []
    for r in sorted(result, key=lambda x: x[3], reverse=True)[:limit]:
        name, content, date, _id = r
        data = {"name": name, "message": content, "time": str(date)}
        results.append(data)

    return list(reversed(results))

def get_messages_by_name(self, name, limit=100):
    """
    Gets a list of messages by user name
    :param name: str
    :return: list
    """
    return self.get_all_messages(limit, name)

def save_message(self, name, msg):
    """
    saves the given message in the table
    :param name: str
    :param msg: str
    :param time: datetime
    :return: None
    """
    query = f"INSERT INTO {PLAYLIST_TABLE} VALUES (?, ?, ?, ?)"
```


filters.py : SE Project/Chatroom/application

```
from jinja2 import Undefined

def _slice(iterable, pattern):
    """
    a custom built slice method that can be used
    inside jinja template engine
    :param iterable: string
    :param pattern: string ex (:::-1)
    :return: string
    """
    if iterable is None or isinstance(iterable, Undefined):
        return iterable

    # convert to list so we can slice
    items = str(iterable)

    start = None
    end = None
    stride = None

    # split pattern into slice components
    if pattern:
        tokens = pattern.split(':')
        print(tokens)
        if len(tokens) > 1:
            start = int(tokens[0])
        if len(tokens) > 2:
            end = int(tokens[1])
        if len(tokens) > 3:
            stride = int(tokens[2])

    return items[start:end:stride]
```

views.py : SE Project/Chatroom/application (1)

```
from flask import Blueprint
from flask import Flask, render_template, url_for, redirect, request, session, jsonify, flash, Blueprint
from .database import DataBase

view = Blueprint("views", __name__)

# GLOBAL CONSTANTS
NAME_KEY = 'name'
MSG_LIMIT = 20

# VIEWS

@view.route("/login", methods=["POST", "GET"])
def login():
    """
    displays main login page and handles saving name in session
    :exception POST
    :return: None
    """
    if request.method == "POST": # if user input a name
        name = request.form["inputName"]
        if len(name) >= 2:
            session[NAME_KEY] = name
            flash(f'You were successfully logged in as {name}.')
            return redirect(url_for("views.home"))
        else:
            flash("Name must be longer than 1 character.")

    return render_template("login.html", **{"session": "session"})

@view.route("/logout")
def logout():
    """
    logs the user out by popping name from session
    :return: None
    """
    session.pop(NAME_KEY, None)
    flash("You were logged out.")
    return redirect(url_for("views.login"))
```

views.py : SE Project/Chatroom/application (2)

```
@view.route("/")
@view.route("/home")
def home():
    """
    displays home page if logged in
    :return: None
    """
    if NAME_KEY not in session:
        return redirect(url_for("views.login"))

    return render_template("index.html", **{"session": session})

@view.route("/history")
def history():
    if NAME_KEY not in session:
        flash("Please login before viewing message history")
        return redirect(url_for("views.login"))

    json_messages = get_history(session[NAME_KEY])
    print(json_messages)
    return render_template("history.html", **{"history": json_messages})

@view.route("/get_name")
def get_name():
    """
    :return: a json object storing name of logged in user
    """
    data = {"name": ""}
    if NAME_KEY in session:
        data = {"name": session[NAME_KEY]}
    return jsonify(data)

@view.route("/get_messages")
def get_messages():
    """
    :return: all messages stored in database
    """
    db = DataBase()
    msgs = db.get_all_messages(MSG_LIMIT)
    messages = remove_seconds_from_messages(msgs)
```

views.py : SE Project/Chatroom/application (3)

```
@view.route("/get_history")
def get_history(name):
    """
    :param name: str
    :return: all messages by name of user
    """
    db = DataBase()
    msgs = db.get_messages_by_name(name)
    messages = remove_seconds_from_messages(msgs)

    return messages

# UTILITIES
def remove_seconds_from_messages(msgs):
    """
    removes the seconds from all messages
    :param msgs: list
    :return: list
    """
    messages = []
    for msg in msgs:
        message = msg
        message["time"] = remove_seconds(message["time"])
        messages.append(message)

    return messages

def remove_seconds(msg):
    """
    :return: string with seconds trimmed off
    """
    return msg.split(".")[0][:-3]
```

index.js. : SE Project/Chatroom/application/static (1)

```

c function add_messages(msg, scroll){
  ( typeof msg.name !== 'undefined' ) {
    var date = dateNow()

    if ( typeof msg.time !== "undefined" ) {
      var n = msg.time
    }else{
      var n = date
    }
    var global_name = await load_name()

    var content = '<div class="container">' + '<b style="color:#000" class="right">' + msg.name + '</b><p>' + msg.message + '</p><span class="time-right">' + n + '</span></div>'
    if (global_name == msg.name){
      content = '<div class="container darker">' + '<b style="color:#000" class="left">' + msg.name + '</b><p>' + msg.message + '</p><span class="time-left">' + n + '</span></div>'
    }
    // update div
    var messageDiv = document.getElementById("messages")
    messageDiv.innerHTML += content

    (scroll){
      scrollSmoothToBottom("messages");
    }
  }
}

c function load_name(){
  return await fetch('/get_name')
    .then(async function (response) {
      return await response.json();
    }).then(function (text) {
      return text["name"]
    });
}

c function load_messages() {
  return await fetch('/get_messages')
    .then(async function (response) {
      return await response.json();
    }).then(function (text) {
      console.log(text)
      return text
    });
}

```

index.js. : SE Project/Chatroom/application/static (2)

```
$(function()
{
    $('.msgs') .css({'height': (($window).height()) * 0.7+'px'});

    $(window).bind('resize', function(){
        $('.msgs') .css({'height': (($window).height()) * 0.7+'px'});
    });
});

function scrollSmoothToBottom (id) {
    var div = document.getElementById(id);
    $('#'+ id).animate({
        scrollTop: div.scrollHeight - div.clientHeight
    }, 500);
}

function dateNow() {
    var date = new Date();
    var aaaa = date.getFullYear();
    var gg = date.getDate();
    var mm = (date.getMonth() + 1);

    if (gg < 10)
        gg = "0" + gg;

    if (mm < 10)
        mm = "0" + mm;

    var cur_day = aaaa + "-" + mm + "-" + gg;

    var hours = date.getHours()
    var minutes = date.getMinutes()
    var seconds = date.getSeconds();

    if (hours < 10)
        hours = "0" + hours;

    if (minutes < 10)
        minutes = "0" + minutes;

    if (seconds < 10)
        seconds = "0" + seconds;

    return cur_day + " " + hours + ":" + minutes;
}
```

index.js. : SE Project/Chatroom/application/static (3)

```

var socket = io.connect('http://' + document.domain + ':' + location.port);
socket.on( 'connect', async function() {
  var usr_name = await load_name()
  if (usr_name != ""){
    socket.emit( 'event', {
      message: usr_name + ' just connected to the server!',
      connect: true
    } )
  }
}
var form = $( 'form#msgForm' ).on( 'submit', async function( e ) {
  e.preventDefault()

  // get input from message box
  let msg_input = document.getElementById("msg")
  let user_input = msg_input.value
  let user_name = await load_name()

  // clear msg box value
  msg_input.value = ""

  // send message to other users
  socket.emit( 'event', {
    message : user_input,
    name: user_name
  } )
} )
} )
/*socket.on( 'disconnect', async function( msg ) {
  var usr_name = await load_name()
  socket.emit( 'event', {
    message: usr_name + ' just left the server...',
  } )
})*
socket.on( 'message response', function( msg ) {
  add_messages(msg, true)
})

window.onload = async function() {
  var msgs = await load_messages()
  for (i = 0; i < msgs.length; i++){
    scroll = false
    if (i == msgs.length-1) {scroll = true}
    add_messages(msgs[i], scroll)
  }

  let name = await load_name()
  if (name != ""){
    $("#login").hide();
  }else{
    $("#logout").hide();
  }
}
}

```


base.html : SE Project/Chatroom/application/templates (1)

```

<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <style type="text/css">
    /* Chat containers */
    .container {
      border: 2px solid #dedede;
      background-color: #f1f1f1;
      border-radius: 5px;
      padding: 10px;
      margin: 10px 0;
    }

    /* Darker chat container */
    .darker {
      border-color: #ccc;
      background-color: #ddd;
    }

    /* Clear floats */
    .container::after {
      content: "";
      clear: both;
      display: table;
    }

    /* Style images */
    .container b {
      float: left;
      max-width: 60px;
      width: 100%;
      margin-right: 20px;
      border-radius: 50%;
    }

    /* Style the right image */
    .container b.right {
      float: right;
      margin-left: 20px;
      margin-right: 0;
    }

    /* Style time text */
    .time-right {
      float: right;
      color: #aaa;
    }

    /* Style time text */
    .time-left {
      float: left;
      color: #999;
    }
  </style>

```


base.html : SE Project/Chatroom/application/templates (2)

```
</style>
<script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-J6qa489bIE+poT4WihKhv5ZF5SrP0B1EJwBVNI7InGFVbhW1yYRrSjOzEn" crossorigin="anonymous"></script>
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/1.7.3/socket.io.min.js"></script>
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vkoo8x4CGSf3HxxBT/jGPQ8tN96ghEqoFf1j9K" crossorigin="anonymous">
</head>

<body>

    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
      <div class="navbar-brand" href="#">CHAT APP</div>
      <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <div class="navbar-nav">
          <a class="nav-item nav-link" href="/home">Home</a> <a class="sr-only" href="#"></a>
          <a class="nav-item nav-link" href="/history">History</a>
          <a class="nav-item nav-link" id="login" href="/login">Login</a>
          <a class="nav-item nav-link" id="logout" href="/logout">Logout</a>
        </div>
      </div>
    </nav>

    {% block messages %}
      {% with messages = get_flashed_messages() %}
        {% if messages %}
          <div class="alert alert-dark" role="alert">
            {% for message in messages %}
              {{ message }}
            {% endfor %}
          </div>
        {% endif %}
      {% endwith %}
    {% endblock %}

    {% block content %}
    {endblock %}

    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6eR8JHqVeR8JHXKHn8Iv7YdZw1w9klDlqLeBeXxpGHyKbp1fPeWxUezbTG2M" crossorigin="anonymous"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-wf2fhqe2gkbakSRJL2FlatDkjNeL7tDfCrkPO1/IvSe/Kvc8/Sepet48/GOP+jI" crossorigin="anonymous"></script>
    <script type="text/javascript" src="{ url_for('static', filename='index.js') }"></script>
  </body>
</html>
```

history.html : SE Project/Chatroom/application/templates

```
{% extends "base.html" %}

{% block title %}Message History{% endblock %}

{% block content %}

<div class="row" style="padding-top:10px;" >
  <div class="col"><hr></div>
  <div class="col-auto"><p class="h3" align="center">View Message History</p></div>
  <div class="col"><hr></div>
</div>
<div class="container">
  <div class="overflow-auto" class="msgs" id="messagesHistory" style="overflow-y: scroll; height:500px;">
    {% for msg in history %}
      <div class="container darker"><b style="color: #000" class="left">{{msg["name"]}}</b><p> {{msg["message"]}}</p><span class="time-left">{{msg["time"]}}</span></div>
    {% endfor %}
  </div>
</div>
{% endblock %}
```

index.html : SE Project/Chatroom/application/templates

```
{% extends "base.html" %}

{% block title %}Home{% endblock %}

{% block messages %}
    {% with messages = get_flashed_messages() %}
        {% if messages %}
            <div class="alert alert-success alert-dismissible fade show" role="alert">
                {% for message in messages %}
                    {{ message }}
                {% endfor %}
                <button type="button" class="close" data-dismiss="alert" aria-label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
            </div>
        {% endif %}
    {% endwith %}
{% endblock %}

{% block content %}
<style>
    .form button:hover,.form button:active,.form button:focus {
        background: #43A047;
    }
</style>
<div class="row" style="padding-top:10px;" >
    <div class="col"><hr></div>
    <div class="col-auto"><p class="h3" align="center">Start Chatting</p></div>
    <div class="col"><hr></div>
</div>
<div class="container">
    <div id="messages" class="overflow-auto msgs" style="overflow-y: scroll; height:500px;">
    </div>
    <form id="msgForm" action="" method="POST" style="bottom:0; margin: 0% 0% 0% 0%;">
        <div class="input-group mb-3">
            <input type="text" class="form-control" placeholder="Message" aria-label="Message" id="msg">
            <div class="input-group-append">
                <button class="btn btn-success" type="submit" id="sendBtn">Send</button>
            </div>
        </div>
    </form>
</div>
{% endblock %}
```

base.html : SE Project/Chatroom/application/templates (1)

```
{% extends "base.html" %}

{% block title %}Login{% endblock %}

{% block messages %}
    {% with messages = get_flashed_messages() %}
        {% if messages %}
            {% for message in messages %}
                {% set display = slice(message, "1:") %}
                {% set prefix = slice[0] %}
                {% if prefix == "1" %}
                    <div class="alert alert-secondary" role="alert">
                        {{ display }}
                    </div>
                {% else %}
                    <div class="alert alert-danger" role="alert">
                        {{ display }}
                    </div>
                {% endif %}
            {% endfor %}
        {% endif %}
    {% endwith %}
{% endblock %}

{% block content %}
<style type="text/css">
@import url(https://fonts.googleapis.com/css?family=Roboto:300);

.login-page {
    width: 360px;
    padding: 8% 0 0;
    margin: auto;
}

.form {
    position: relative;
    z-index: 1;
    background: #FFFFFF;
    max-width: 360px;
    margin: 0 auto 100px;
    padding: 45px;
    text-align: center;
    box-shadow: 0 0 20px 0 rgba(0, 0, 0, 0.2), 0 5px 5px 0 rgba(0, 0, 0, 0.24);
}
```

base.html : SE Project/Chatroom/application/templates (2)

```

.form input {
    font-family: "Roboto", sans-serif;
    outline: 0;
    background: #f2f2f2;
    width: 100%;
    border: 0;
    margin: 0 0 15px;
    padding: 15px;
    box-sizing: border-box;
    font-size: 14px;
}

.form button {
    text-transform: uppercase;
    outline: 0;
    width: 100%;
    border: 0;
    padding: 15px;
    color: #FFFFFF;
    font-size: 14px;
    -webkit-transition: all 0.3 ease;
    transition: all 0.3 ease;
    cursor: pointer;
}

.form button:hover, .form button:active, .form button:focus {
    background: #43A047;
}

.form .message {
    margin: 15px 0 0;
    color: #b3b3b3;
    font-size: 12px;
}

.container {
    position: relative;
    z-index: 1;
    max-width: 300px;
    margin: 0 auto;
}

.container:before, .container:after {
    content: "";
    display: block;
    clear: both;
}

</style>

<div class="login-page">
    <div class="form">
        <form class="login-form" action="#" method="POST">
            <input type="text" name="inputName" id="inputName" placeholder="Your Name"/>
            <button type="submit" class="btn btn-success mb-2">Submit</button>
        </form>
    </div>
</div>

<script type="text/javascript" src="{% url_for('static', filename='index.js') %}"></script>
{% endblock %}

```

Results and Discussion

Limitations

There are mainly two limitations of the project and that are:

- The firewall is to be disabled for intra network.
- It is dependent on the specific algorithm used.

Future Enhancements

There is always a room for improvements in any software package, however good and efficient it may be done. But the most important thing should be flexible to accept further modification. Right now we are just dealing with text communication. In future this software may be extended to include features such

as:

File transfer: this will enable the user to send files of different formats to others via the chat

application.

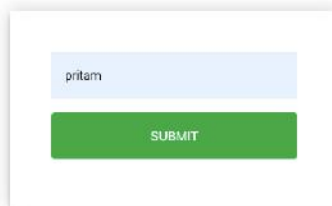
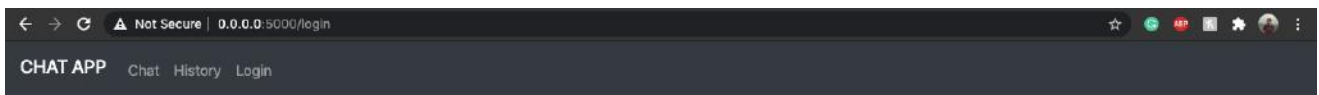
Voice chat: this will enhance the application to a higher level where communication will be

possible via voice calling as in telephone.

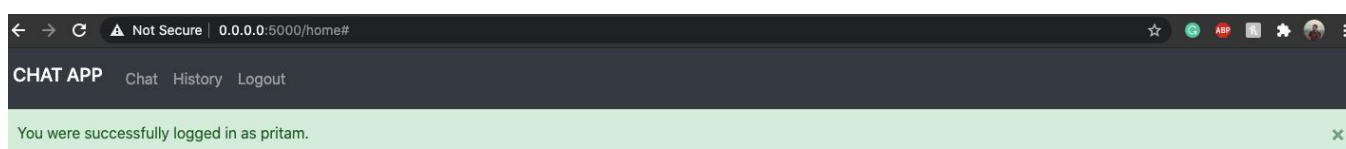
. Video chat: this will further enhance the feature of calling into video communication.

Complete Execution Screenshots

Login page:

A login form displayed on a white background. It consists of a light blue rectangular text input field containing the text 'pritam'. Below the input field is a solid green rectangular button with the word 'SUBMIT' written in white capital letters.

Chat Window



Start Chatting

pritam hello

2020-10-28 14:10

pritam hello

2020-10-28 14:13

hey there

rishi
2020-10-28 14:13

how you are

rishi
2020-10-28 14:13

pritam h

2020-10-28 14:20

pritam h

2020-10-28 14:20

Message

Send

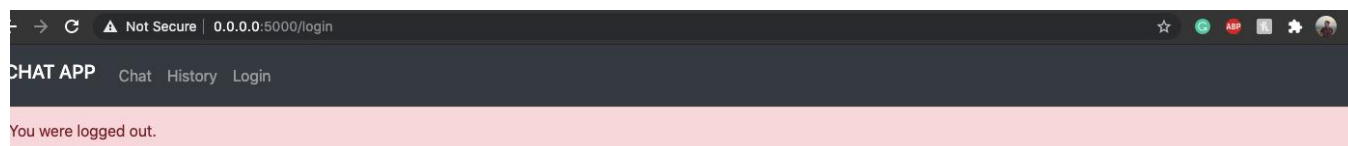
History



View Message History

pritam	hello	2020-10-28 14:10
pritam	hello	2020-10-28 14:13
pritam	h	2020-10-28 14:20
pritam	h	2020-10-28 14:20
pritam	h	2020-10-28 14:20

Logout



SUBMIT