A

PROJECT REPORT

ON

# EMOJIFY


**By**

VIDIT SHAH (CE-03) (19CEUON075)

CHAVDA BHAVIN (CE-23) (19UOG024)


**B.Tech CE Semester-VI**

**Subject: SYSTEM DESIGN PRACTICE (CE-612)**

**Guided by:**

Prof. Shaifali P. Malukani (Assistant Professor)

Dept. of Computer Engg.

**Faculty of Technology Department of Computer Engineering**

**Dharmsinh Desai University**

# CERTIFICATE

This is to certify that the practical / term work carried out in the subject of **SYSTEM DESIGN PRACTICE (CE-612)** and recorded in this journal is the bonafide work of

VIDIT SHAH (CE-03) (19CEUON075)

CHAVDA BHAVIN (CE-23) (19UOG024)

of B.Tech Semester **VI** in the branch of **Computer Engineering** during the academic year **2021-2022**.

Project Guide:

Prof. Shaifali P. Malukani

Assistant Professor

Dept. of Computer Engg.,

Faculty of Technology

Dharmsinh Desai University, Nadiad

# INDEX

# 1. Abstract

Chatting is now-a-days very useful to express our ideas as well as receive others ideas on any topic. Chats reflect the recent trends of the society. Sometimes, it is possible to meet eminent people in chatting and have their advice. So, we bring chat application for our users with great feature such as emojify which recognizes users face expression and prints emoji. The combination of visual and textual content in the same message builds up a modern way of communication.

# 2. Introduction

## 2.1 Brief Introduction

In the fast-moving world, most of us are connected with chatting. User can create group and connect with multiple people at same time. Chat and interact with other members of the group. Admin can edit/remove member of the group. As Emojis have become a new language that can more effectively express an idea or emotion we bring seven different emotions recognize of user and giving you the best and most accurate result in form of emoji.

Emojify- We are building a convolution neural network to recognize facial emotions. We will be training our model on the FER2013 dataset. Then we'll map those emotions with the corresponding emojis. Fer2013 contains approximately 30,000 facial RGB images of different expressions with size restricted to 48×48, and the main labels of it can be divided into 7 types: 0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral.

After training 4 models, we found a model with better accuracy as we added more images to data of ourselves and tested it with the data.

## 2.2 Tools/Technologies Used

### Technologies:

- Python
- MongoDB
- Flask
- HTML, CSS, JS
- JQuery
- Socket io
- Neural Network

### Tools:

- Visual Studio Code
- MongoDB Atlas
- Github

### Platform:

- Local development server

### Libraries:

- Numpy
- Pandas
- Tensorflow
- Keras
- matplotlib

# 3. Software Requirement Specifications

## 3.1 Product Scope:

This System is designed to connect and share their real time emotion through emojis with each other.

## 3.2 System Functional Requirements

### R.1 Manage User

#### R.1.1 Sign up

Description: User creates account by entering details.

Input: Username, email address and password.

Output: Login page is displayed.

Processing: If username is unique then user is transferred to login page.

#### R.1.1 Login

Description: User logins to the system by entering valid username and password.

Input: Username and Password.

Output: Index Page is displayed having chat rooms if user successfully logs in.

Processing: Username and Password validation.

### R.2 Manage Room

#### R.2.1 Create Room

Description: User can create new room and add his friends to it.

Input: Room name, Friend's username

Output: Room is successfully created and displayed in my rooms.

R.2.2 Edit Room

Description: Admin can edit room name and add new member.

Input: Room name, Member username

Output: Room name is changed and new member is added to the room.

**R.3 Manage Message:**

R.3.1 Send Message

Description: Type your message in box and send.

Input: Message

Output: Message is sent successfully and other members can read the message.

**R.4 Manage Emotion:**

R.4.1 Recognize Facial Expression

Description: Recognize facial expression and maps it with emojis.

Input: Opens camera and take face expression

Output: Emoji is printed in chat box.

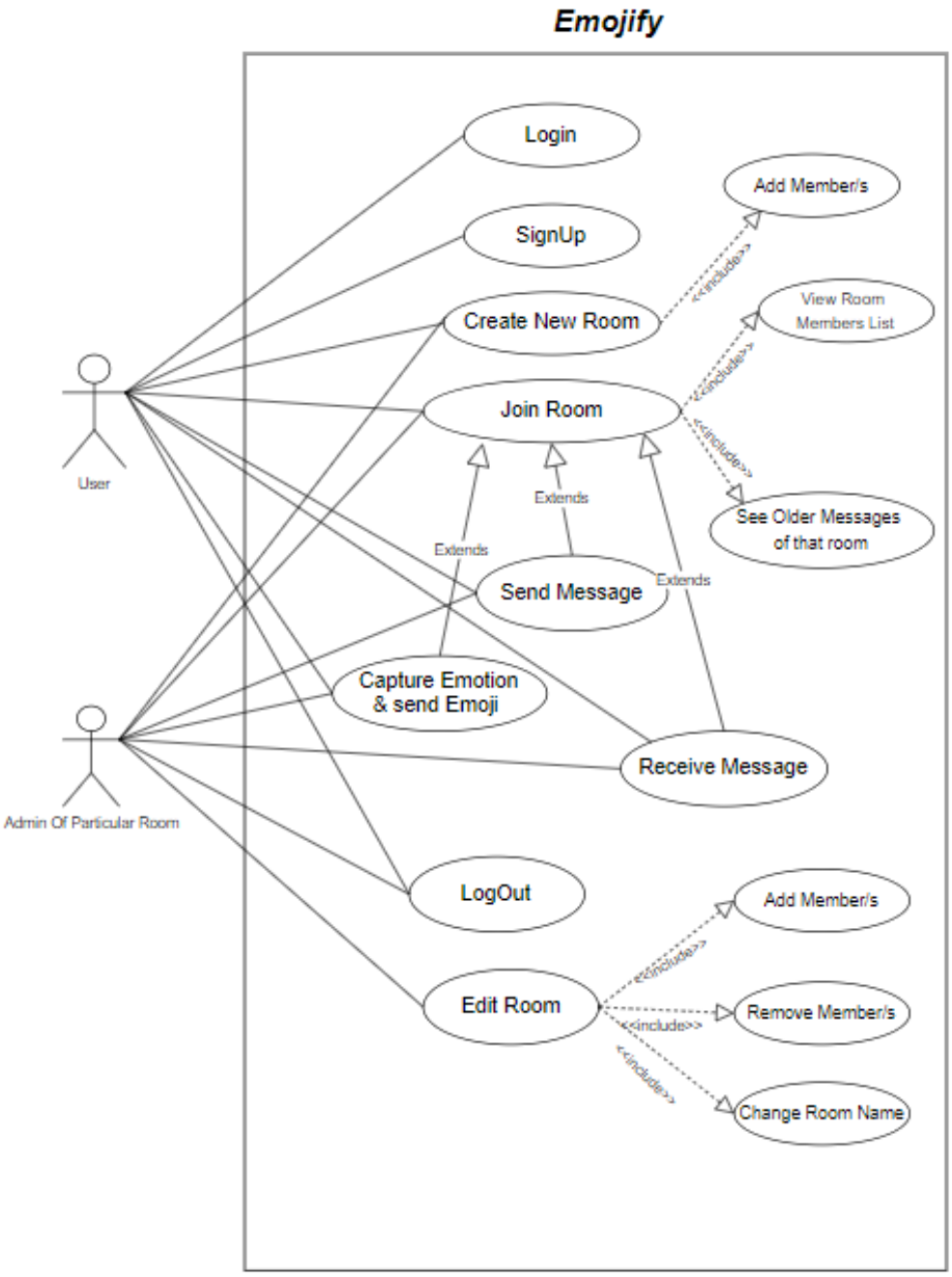
## 3.3 Other Non-functional Requirements:

1. Performance:
   The system must be interactive and must not involve long delays. Though in case of opening the website components or loading the page the system shows the delay less than 2 seconds.
2. Safety:
   The user's data is highly personal. The system has authorization to avoid any un-authorization access to user's private data.

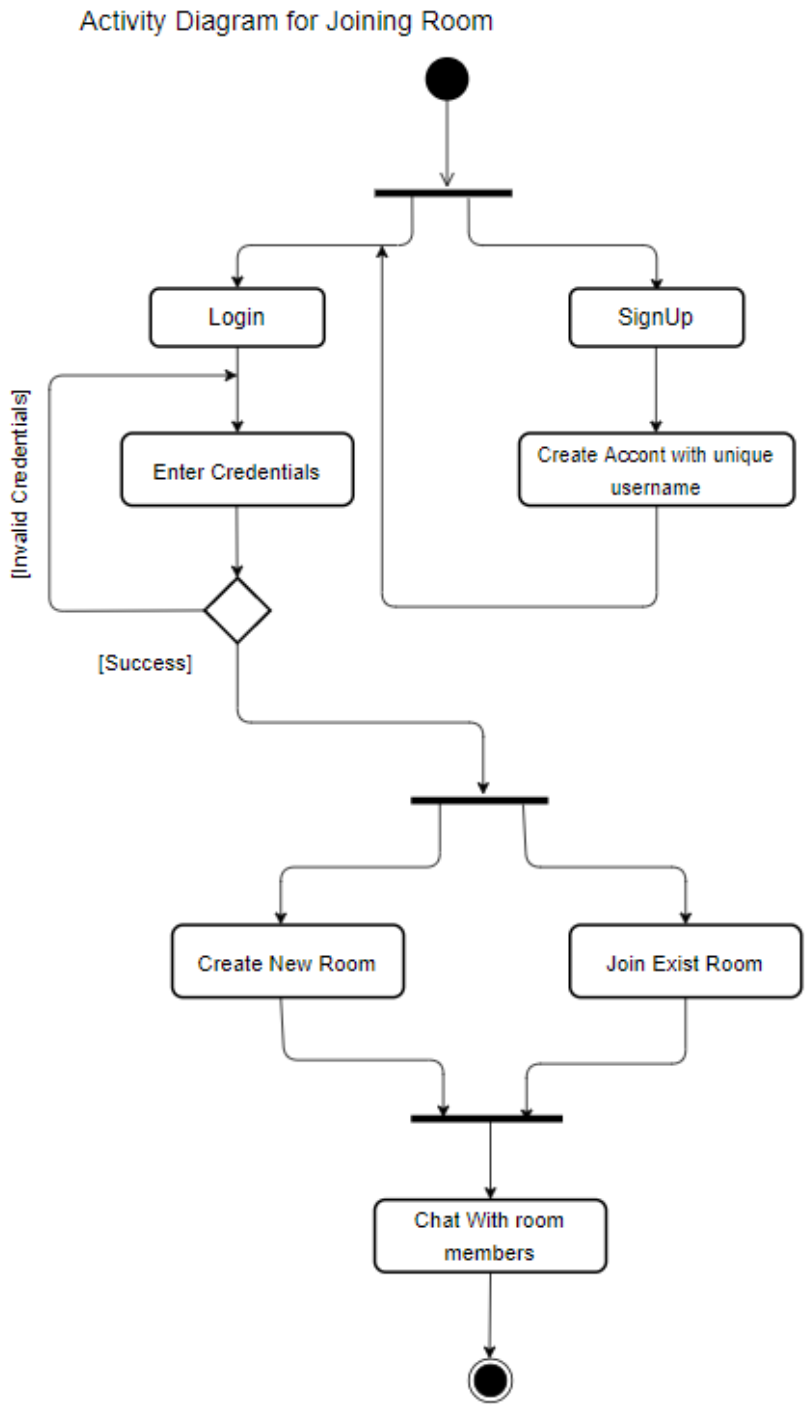
3. Reliability:
   As the system has personal data, its reliability is the major factor for consideration.

# 4. Design

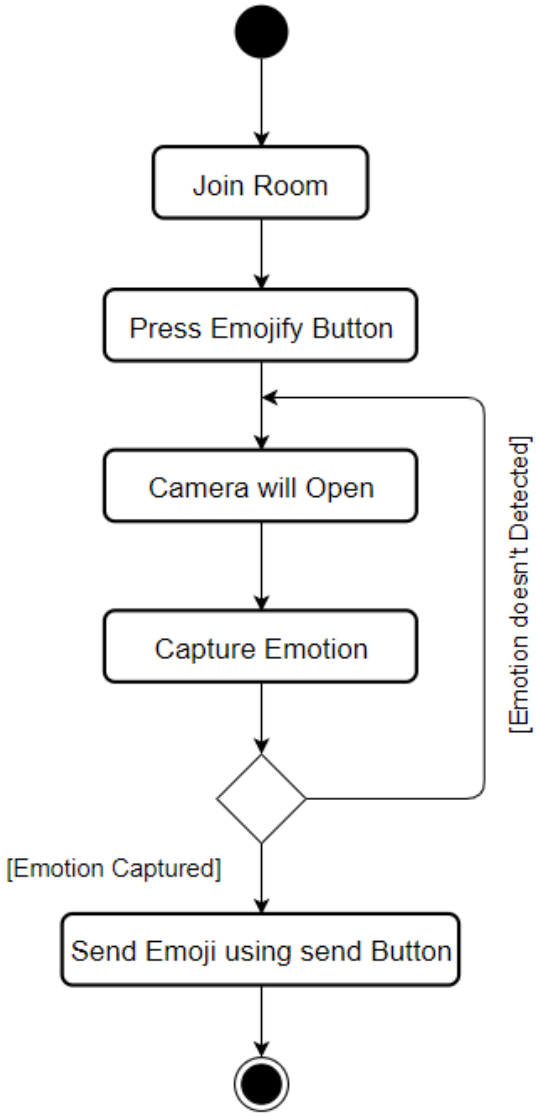
## 4.1 Use Case Diagram:



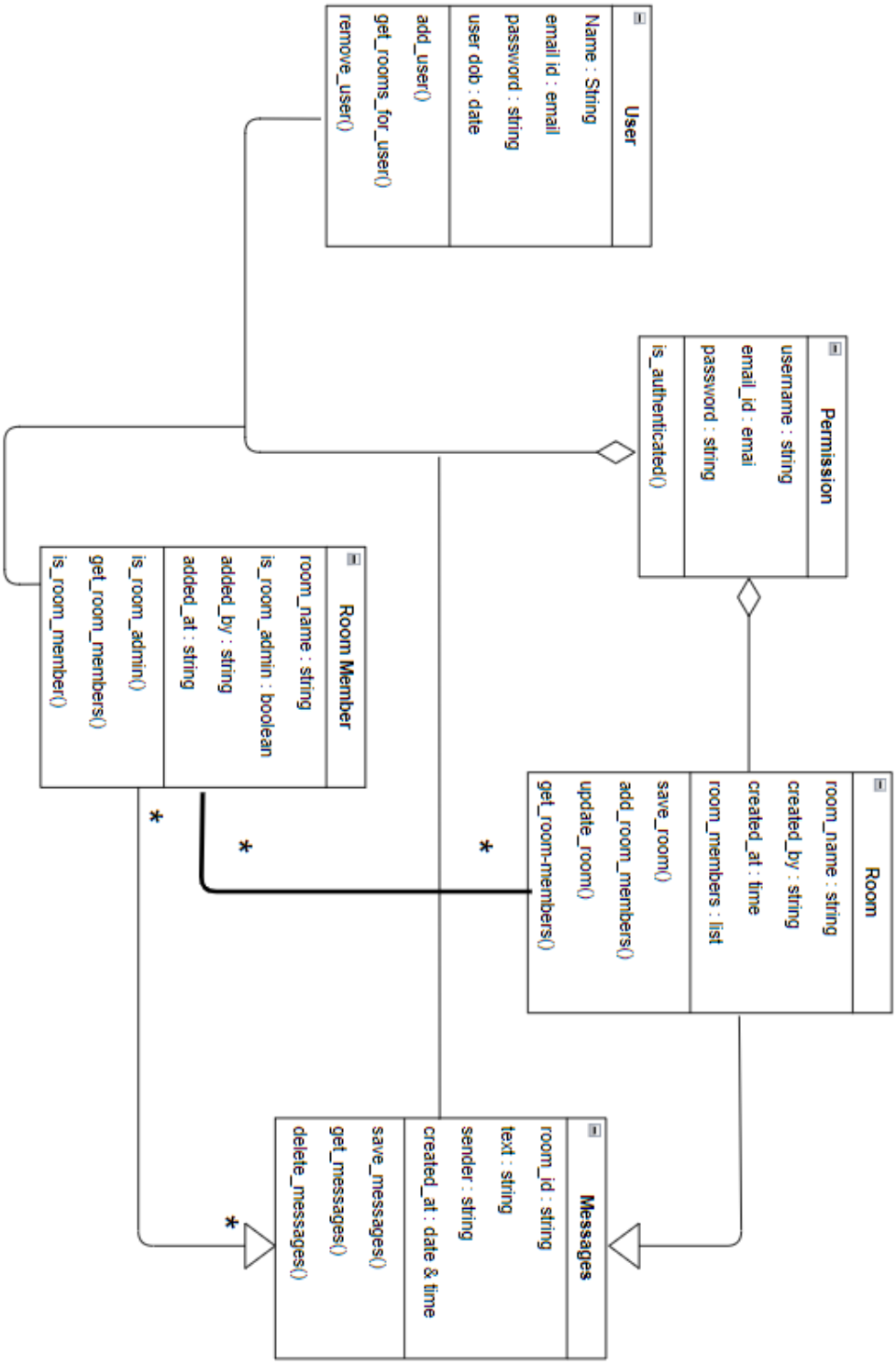
## 4.2 Activity Diagram:



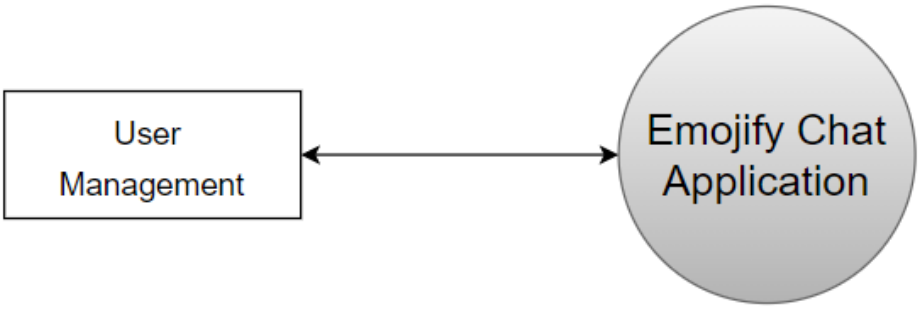
Send Emoji using Emojify button



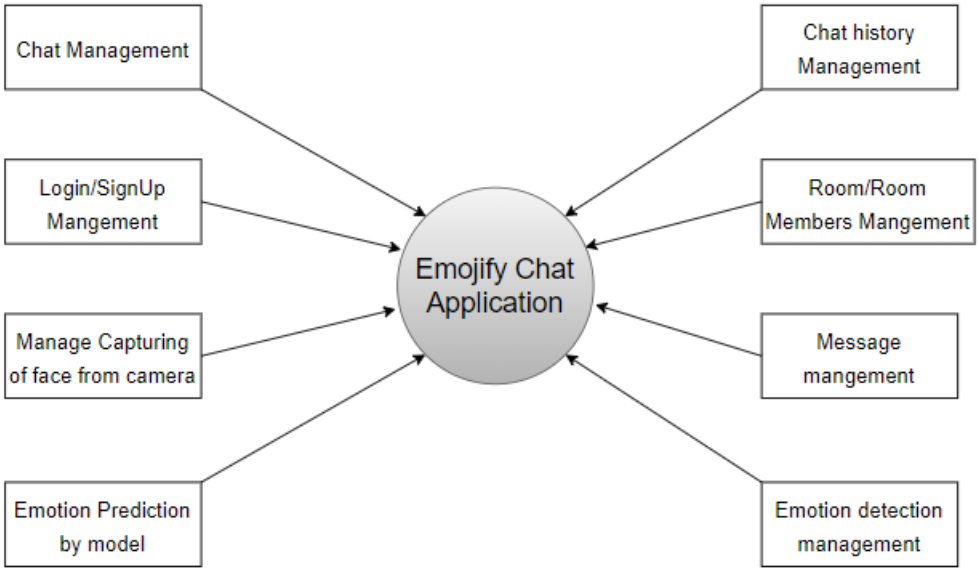
## 4.3 Class Diagram:



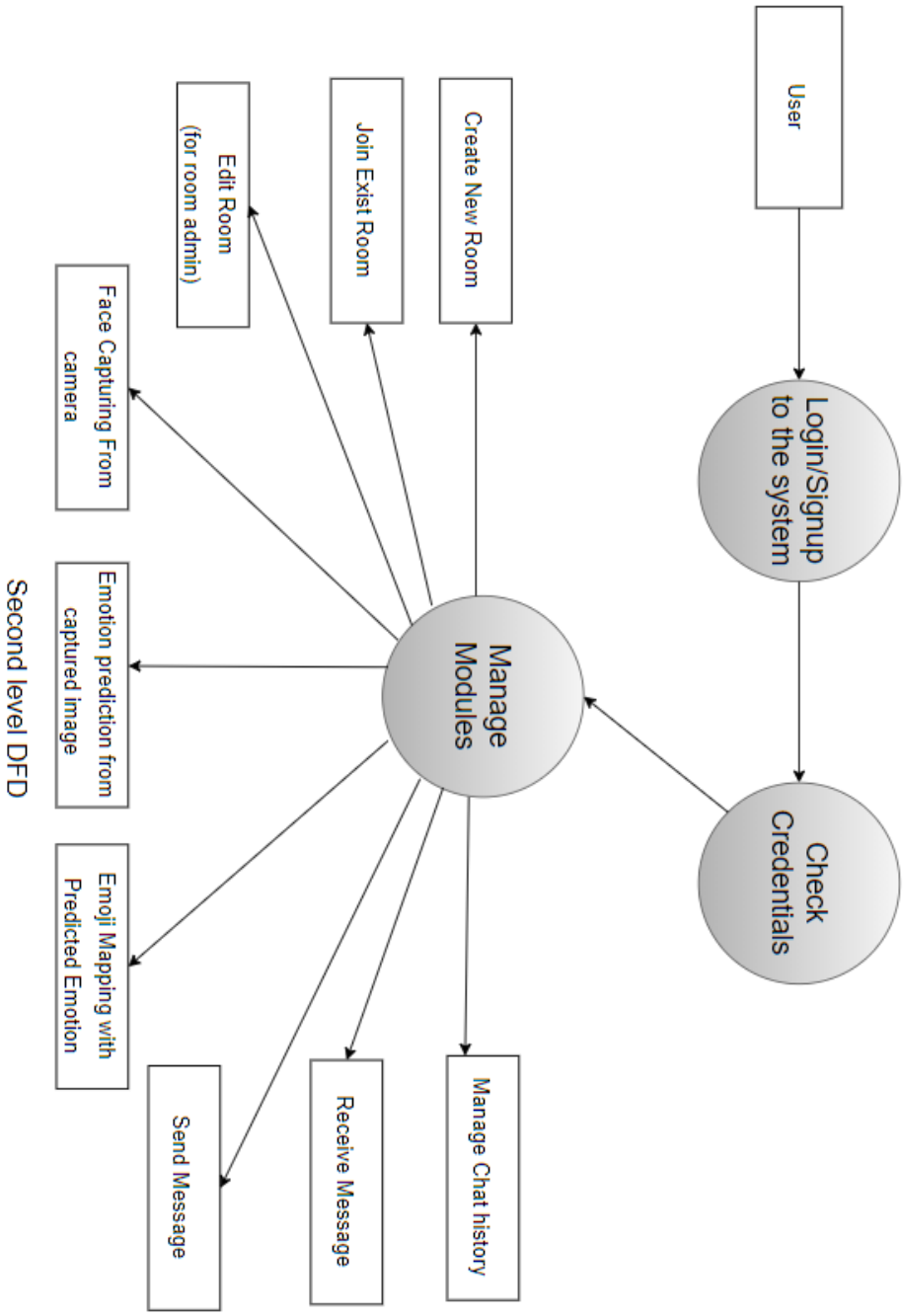
## 4.4 DFD Model:




Zero level DFD

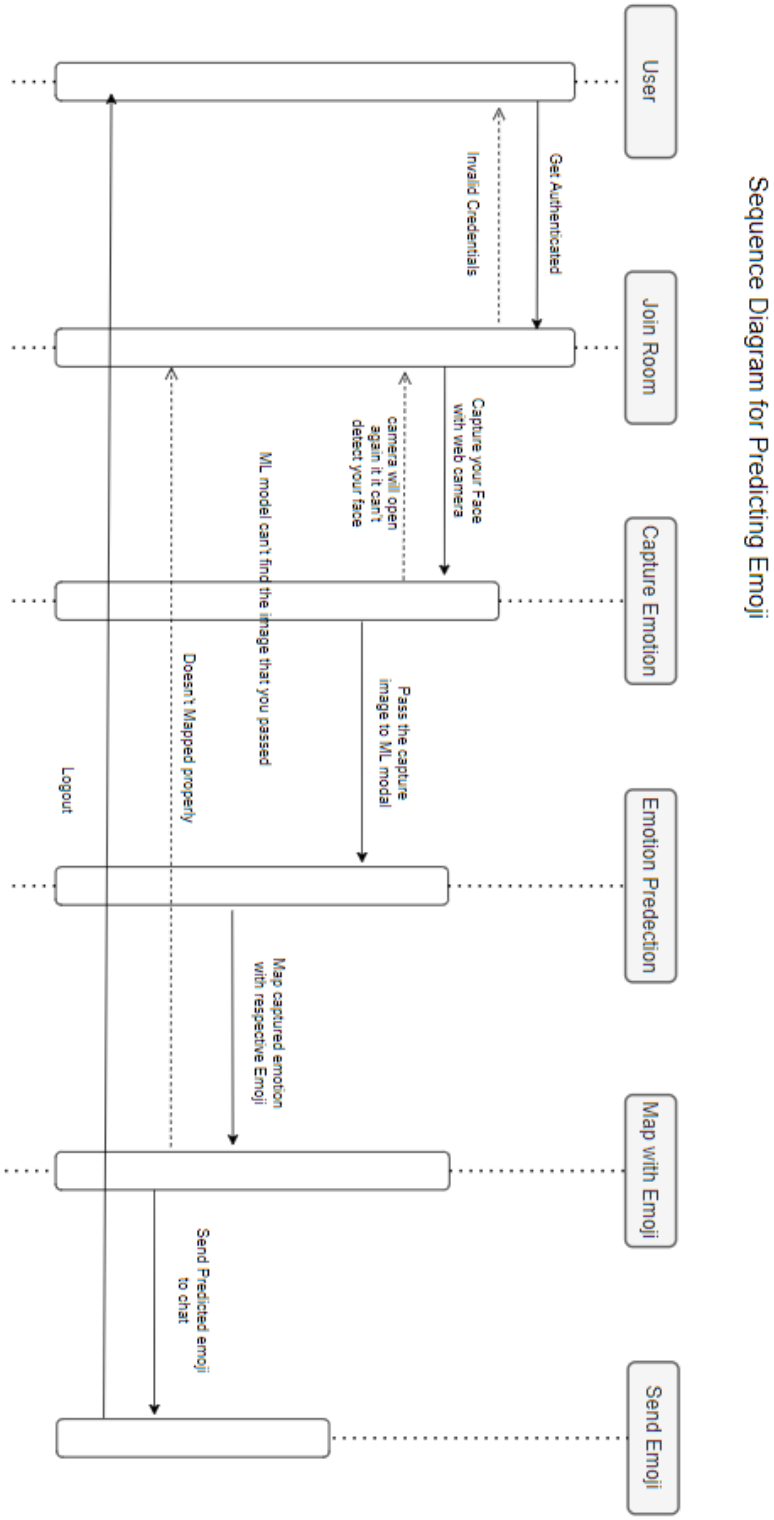



First level DFD




Second level DFD

## 4.5 Sequence Diagram:



# 5. Implementation

The system consists of 4 basic modules namely

1. Login Module
2. Room Module
3. Chat Module
4. Emoji Module

## 5.1 Login Module:

User will login using his/her credentials and will be redirected to Index Page.

## 5.2 Room Module:

User can create room and join the existing rooms. If user is admin of room, he/she can edit room participant.

## 5.3 Chat Module:

User can send, receive and load older messages. Also, he/she can view members in the group.

## 5.4 Emoji Module:

Capture users face expression and gives an emoji in the text box.

## 5.5 Function Prototypes:

Login Page:

```
<body>
    <h1>Login Here</h1>
    <h3 class="message">{{message}}</h3>
    <form action="/login" , method="post">
        <label>Username : </label>
        <input type="text" , name="username" placeholder="Enter Your Username">
        <br>
        <label>Password : </label>
        <input type="password" , name="password" placeholder="Enter Your Password">
        <br>
        <button type="submit">Login</button>
    </form>
    <h3 class="sgup">New To Our Emojify Chat Application , SignUp below</h3><br>
    <a href="/signup" class="signupbtn">Sign Up</a>
</body>
```

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('home'))

    message = ''
    if request.method=='POST':
        username = request.form.get('username')
        password_input =request.form.get('password')
        user = get_user(username)

        if user and user.check_password(password_input):
            login_user(user)
            return redirect(url_for('home'))
        else:
            message = 'Invalid Credentials, Please try again'

    return render_template('login.html' , message=message)
```

## Room Module:

```
{% if current_user.is_authenticated %}
<h3>My rooms</h3>
<ul>
    {% for room in rooms %}
    <!-- {{room._id.room_id}} -->
    <li><a href="/rooms/{{room._id.room_id}}">{{ room.room_name }}</a></li>
    {% endfor %}
</ul>
<h3>Want Create Your Own Room , Visit below Link</h3>
<a href="/create-room/">Create-New-room</a>
<h3>Logout Here</h3>
<a href="/logout">Logout</a>
{% endif %}
```

```
<body>
    <h1>Create Room</h1>
    <h3>{{message}}</h3>
    <form action="" method="post">
        <label>Room Name : </label>
        <input type="text" name="room_name">
        <label>Members : </label>
        <input type="text" name="members">
        <br>
        <button type="submit">Submit</button>
    </form>
</body>
```

```
@app.route('/create-room/' , methods=['GET' , 'POST'])
@login_required
def create_room():
    message = ''
    if request.method == 'POST':
        room_name = request.form.get('room_name')
        usernames = [username.strip() for username in request.form.get('members').split(',')]

        if len(room_name) and len(usernames):
            room_id = save_room(room_name, current_user.username)
            if current_user.username in usernames:
                usernames.remove(current_user.username)
            add_room_members(room_id, room_name, usernames, current_user.username)
            return redirect(url_for('view_room', room_id=room_id))
        else:
            message = "Failed to create room"
    return render_template('create_room.html', message=message)
```

```python
@socketio.on('join_room')
def handle_join_room_event(data):
    app.logger.info("{} has joined the room {}".format(data['username'] , data['room']))
    join_room(data['room'])
    socketio.emit('join_room_announcement' , data)
```

```html
<body>
    <h1>Edit Room</h1>
    <h3>{{ message }}</h3>
    <form method="post">
        <label>Room Name:</label>
        <input type="text" name="room_name" value="{{ room.name }}">
        <br>
        <label>Members:</label>
        <input type="text" name="members" value="{{ room_members_str }}">
        <br>
        <button type="submit">Submit</button>
    </form>
</body>
```

```python
@app.route('/rooms/<room_id>/edit', methods=['GET', 'POST'])
@login_required
def edit_room(room_id):
    room = get_room(room_id)
    if room and is_room_admin(room_id, current_user.username):
        existing_room_members = [member['_id']['username'] for member in get_room_members(room_id)]
        room_members_str = ",".join(existing_room_members)
        message = ''
        if request.method == 'POST':
            room_name = request.form.get('room_name')
            room['name'] = room_name
            update_room(room_id, room_name)

            new_members = [username.strip() for username in request.form.get('members').split(',')]
            members_to_add = list(set(new_members) - set(existing_room_members))
            members_to_remove = list(set(existing_room_members) - set(new_members))
            if len(members_to_add):
                add_room_members(room_id, room_name, members_to_add, current_user.username)
            if len(members_to_remove):
                remove_room_members(room_id, members_to_remove)
            message = 'Room edited successfully'
            room_members_str = ",".join(new_members)
        return render_template('edit_room.html', room=room, room_members_str=room_members_str, message=message)
    else:
        return "Room not found", 404
```

## Chat Module:

```
<body>
    <p class="welcome">Welcome {{username}} To {{room.name}}</p>

    <div class="main">
    <div class="left">
        <div id="messages">
            {% for message in messages %}
            <div class="message">
                <div class="main-msg">
                    <b class="own">{{message.sender}}</b>
                    <h3 class="text">{{message.text}}</h3>
                </div>
                <span class="time">{{message.created_at}}</span>
            </div>
            {% endfor %}
        </div>
        <div class="input">
            <form id="emoji_form">
                <button class="submit" type="submit"><img class="input-icon" src="/./static/icons/smiling-face.png"/></button>
            </form>
            <form id="message_input_form">
                <input type="text" id="message_input" placeholder="Enter Your Message">
                <button class="submit" type="submit"><img class="input-icon" src="/./static/icons/send.png"/></button>
            </form>
        </div>
    </div>
```

```
<script type="text/javascript" charset="utf-8">
    var socket = io();
    socket.on('connect', function () {
        socket.emit('join_room', {
            username: "{{ username }}",
            room: "{{room._id}}"
        })
        let message_input = document.getElementById('message_input');
        document.getElementById('message_input_form').onsubmit = function (e) {
            e.preventDefault();
            let message = message_input.value.trim()
            if (message.length) {
                socket.emit('send_message', {
                    username: "{{ username }}",
                    room: "{{room._id}}",
                    message: message
                })
            }
            message_input.value = '';
            message_input.focus();
        }
        document.getElementById('emoji_form').onsubmit = function (e) {
            e.preventDefault();
            socket.emit('emotion')
        }
```

```python
@socketio.on('send_message')
def handle_send_message_event(data):
    app.logger.info("{} has sent message to the room {} : {}".format(data['username'] , data['room'] , data['message']))
    data['created_at'] = datetime.now().strftime("%d %b, %H:%M")
    save_message(data['room'] , data['message'] , data['username'])
    socketio.emit('receive_message' , data , room=data['room'])
```

```javascript
socket.on('receive_message', function (data) {
    console.log(data);
    const newNode = document.createElement('div');
    newNode.innerHTML = `<div class="message">
                            <div class="main-msg">
                                <b class="own">${data.username}</b>
                                <h3 class="text">${data.message}</h3>
                            </div>
                            <span class="time">${data.created_at}</span>
                        </div>`;
    document.getElementById('messages').appendChild(newNode);
    document.getElementById('message_input').value = ''
})
```

```html
<div class="right">
    <button id="load_older_messages_btn">Load older Messages</button>
    <div class="main_member">
        <img class="member_icon" src="/../static/icons/people.png" alt="people"/>
        <h2 class="mem">Members: </h2>
    </div>
    <ul class="members">
        {% for member in room_members%}
        <li class="member">{{member._id.username}}</li>
        {% endfor %}
    </ul>
```

```javascript
document.getElementById("load_older_messages_btn").onclick = (e) => {
    page += 1;
    fetch("/rooms/{{ room._id }}/messages?page=" + page, {
        method: 'GET',
        headers: {
            'Content-Type': 'application/json'
        }
    }).then(response => {
        response.json().then(messages => {
            messages.reverse().forEach(message => prepend_message(message.text, message.sender, message.created_at));
        })
    })
};
```

```python
MESSAGE_FETCH_LIMIT = 3

def get_messages(room_id, page=0):
    offset = page * MESSAGE_FETCH_LIMIT
    messages = list(
        messages_collection.find({'room_id': room_id}).sort('_id', DESCENDING).limit(MESSAGE_FETCH_LIMIT).skip(offset))
    for message in messages:
        message['created_at'] = message['created_at'].strftime("%d %b, %H:%M")
    return messages[::-1]
```

## Emoji Module:

```
model = load_model('fer_final.h5')
face_haar_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

def livevideo():
    cam = cv2.VideoCapture(0)

    cv2.namedWindow("test")

    while True:
        ret, frame = cam.read()
        if not ret:
            print("failed to grab frame")
            break
        cv2.imshow("test", frame)

        k = cv2.waitKey(1)
        if k % 256 == 27:
            # ESC pressed
            print("Escape hit, closing...")
            break
        elif k % 256 == 32:
            gray_img= cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

            faces_detected = face_haar_cascade.detectMultiScale(gray_img, 1.32, 5)
```

```
for (x,y,w,h) in faces_detected:
    print('WORKING')
    cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),thickness=7)
    roi_gray=gray_img[y:y+w,x:x+h]          #cropping region of interest i.e. face area from  image
    roi_gray=cv2.resize(roi_gray,(48,48))
    img_pixels = image.img_to_array(roi_gray)
    img_pixels = np.expand_dims(img_pixels, axis = 0)
    img_pixels /= 255
    predictions = model.predict(img_pixels)

    #find max indexed array

    max_index = np.argmax(predictions[0])
    # max_index=predictions[0]
    if (max_index==0 or max_index==1 or max_index==2 or max_index==3 or max_index==4 or max_index==5 or max_index==6):
        print(max_index)
        cam.release()
        return max_index
```

```
@socketio.on('emotion')
def emotion_handle():
    max_index=livevideo()
    label_map =['😡','🤢', '😨' , '🙂' , '😐', '😔' , '😮' ]
    final = label_map[max_index]

    socketio.emit('catch_emotion' , final)
```

```
socket.on('catch_emotion', function (final) {
    console.log(final)
    let append_emoji = final
    document.getElementById('message_input').value += append_emoji
    document.getElementById('message_input').focus()
})
```

## 5.6 Model Training-

### 1. Importing Libraries:

[2]:

```
import numpy as np
import pandas as pd
import tensorflow as tf
import os
from keras.preprocessing.image import ImageDataGenerator, load_img
from keras.layers import Conv2D, Dense, BatchNormalization, Activation, Dropout, MaxPooling2D, Flatten
from tensorflow.keras.optimizers import Adam, RMSprop, SGD
from keras.callbacks import ModelCheckpoint,EarlyStopping
import datetime
from keras import regularizers
import matplotlib.pyplot as plt
from keras.utils.vis_utils import plot_model
```

### 2. Loading data:

[3]:

```
train_dir = '../input/final12/train/train/'
test_dir = '../input/final12/test/test/'

row, col = 48, 48
classes = 7

def count_exp(path, set_):
    dict_ = {}
    for expression in os.listdir(path):
        dir_ = path + expression
        dict_[expression] = len(os.listdir(dir_))
    df = pd.DataFrame(dict_, index=[set_])
    return df
train_count = count_exp(train_dir, 'train')
test_count = count_exp(test_dir, 'test')
print(train_count)
print(test_count)
```

```
       surprise  fear  angry  neutral   sad  disgust  happy
train      3180  4109   4012     4971  4853      444   7225
      surprise  fear  angry  neutral   sad  disgust  happy
test       840  1032    971     1239  1263      119   1784
```

3. Data Augmentation:
   rescale = to scale down the pizel values in our image between 0 and 1.
   horizontal_flip = flips the image horizontally.
   validation_split = reserves some images to be used for validation purpose.

[5]:
```
train_datagen = ImageDataGenerator(rescale=1./255,
                                   horizontal_flip=True,
                                   validation_split=0.2)

training_set = train_datagen.flow_from_directory(train_dir,
                                                 batch_size=64,
                                                 target_size=(48,48),
                                                 shuffle=True,
                                                 color_mode='grayscale',
                                                 class_mode='categorical',
                                                 subset='training')
validation_set = train_datagen.flow_from_directory(train_dir,
                                                 batch_size=64,
                                                 target_size=(48,48),
                                                 shuffle=True,
                                                 color_mode='grayscale',
                                                 class_mode='categorical',
                                                 subset='validation')

test_datagen = ImageDataGenerator(rescale=1./255,
                                   horizontal_flip=True)
test_set = test_datagen.flow_from_directory(test_dir,
                                                 batch_size=64,
                                                 target_size=(48,48),
                                                 shuffle=True,
                                                 color_mode='grayscale',
                                                 class_mode='categorical')
```

```
Found 23038 images belonging to 7 classes.
Found 5756 images belonging to 7 classes.
Found 7248 images belonging to 7 classes.
```

## 4. Modelling

[7]:

```
weight_decay = 1e-4

num_classes = 7

model = tf.keras.models.Sequential()

model.add(Conv2D(64, (4,4), padding='same', kernel_regularizer=regularizers.l2(weight_decay), input_shape=(48,48,1)))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (4,4), padding='same', kernel_regularizer=regularizers.l2(weight_decay)))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))

model.add(Conv2D(128, (4,4), padding='same', kernel_regularizer=regularizers.l2(weight_decay)))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.3))

model.add(Conv2D(128, (4,4), padding='same', kernel_regularizer=regularizers.l2(weight_decay)))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (4,4), padding='same', kernel_regularizer=regularizers.l2(weight_decay)))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
```

```
model.add(Dropout(0.4))
model.add(Flatten())
model.add(Dense(128, activation="linear"))
model.add(Activation('elu'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer=Adam(0.0003), metrics=['accuracy'])

model.summary()
```

## 5. Training the model

[9]:

```
steps_per_epoch = training_set.n // training_set.batch_size
validation_steps = validation_set.n // validation_set.batch_size

history = model.fit(x=training_set,
                    validation_data=validation_set,
                    epochs=200,
                    callbacks=[checkpointer],
                    steps_per_epoch=steps_per_epoch,
                    validation_steps=validation_steps)
```

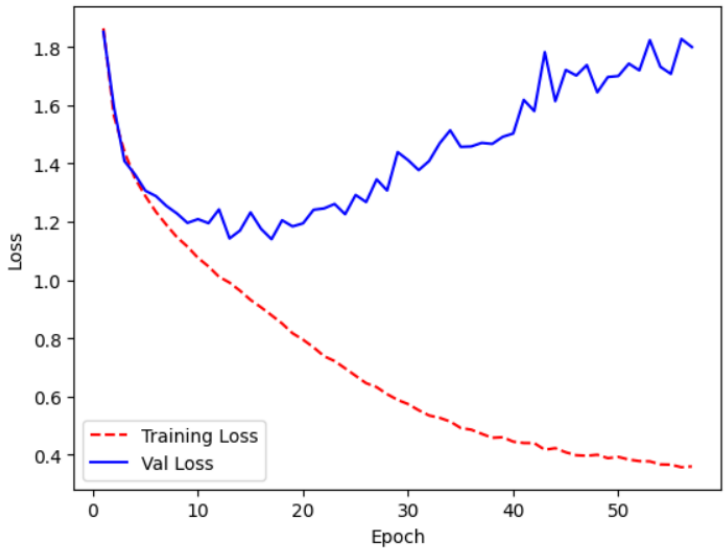
## 6. Saving the model

[12]:

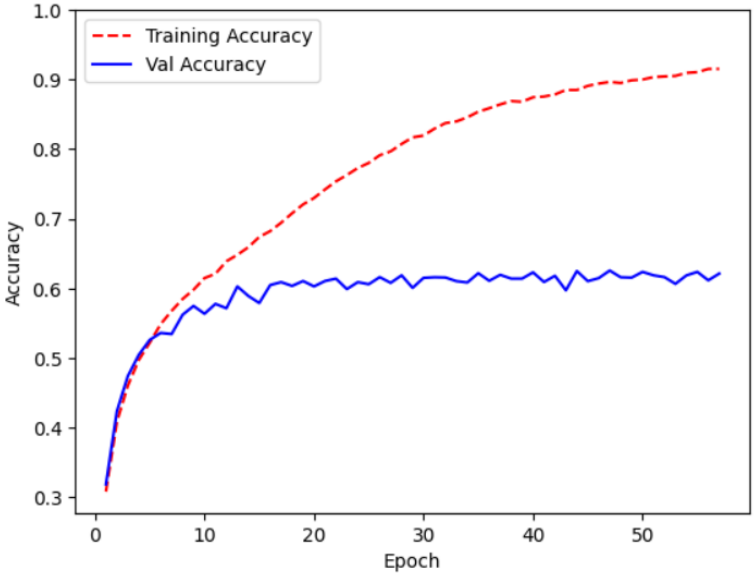
```
model.save("fer_final.h5")
```

# 6. Testing

## Training Loss:



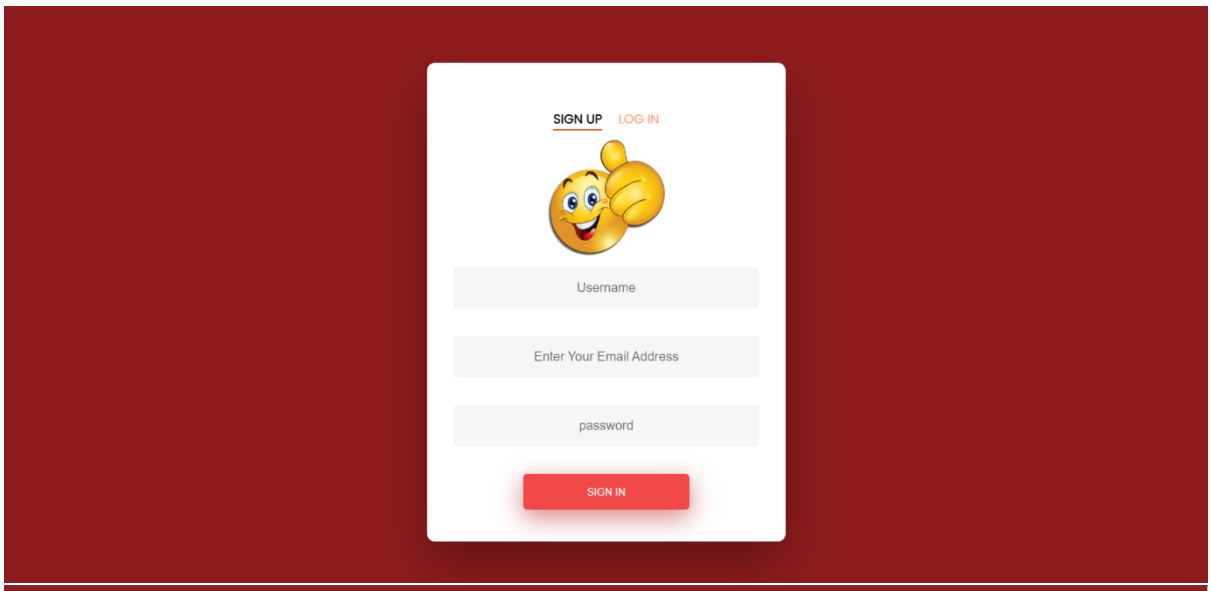

## Training Accuracy Achieved:

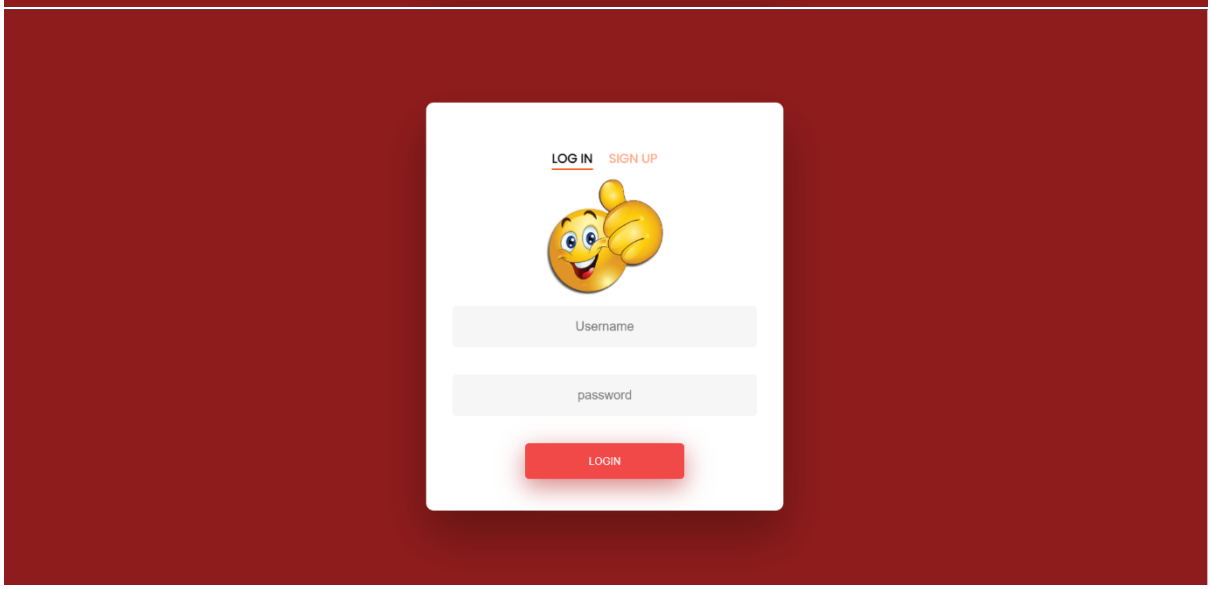


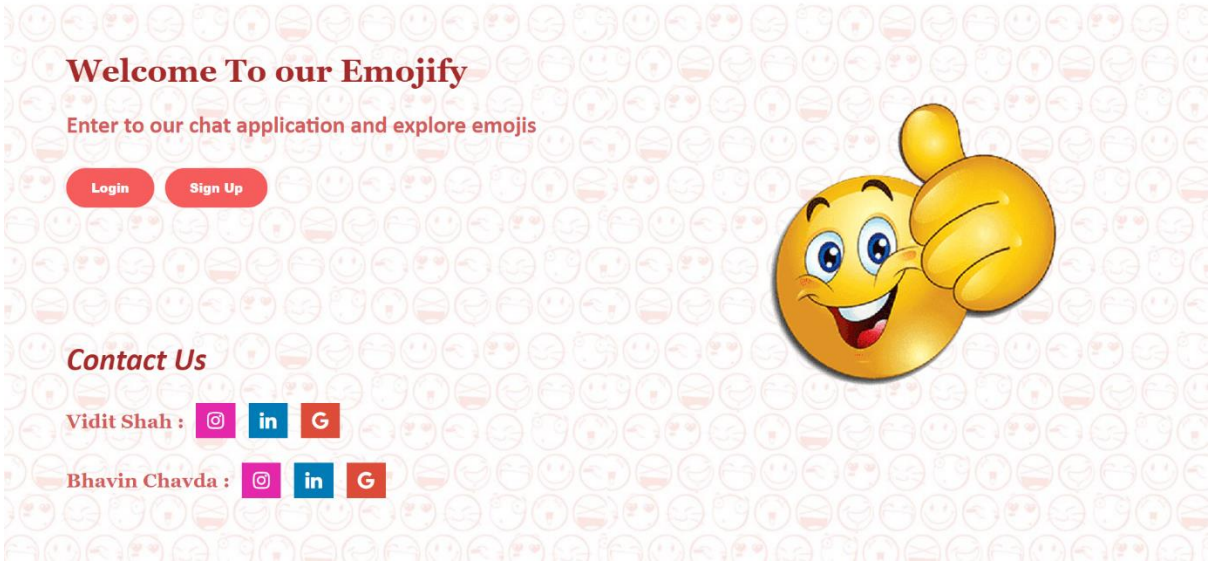

## Test Accuracy:

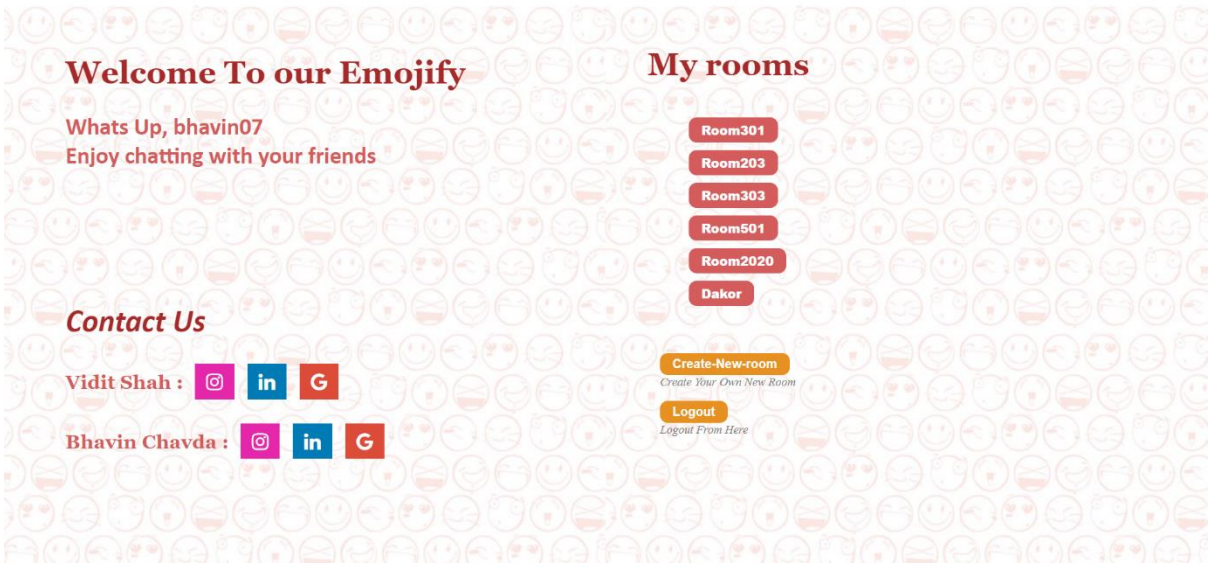
```
113/113 [==============================] - 16s 141ms/step - loss: 1.7293 - accuracy: 0.6225
Test accuracy = 62.2511088848114%
```

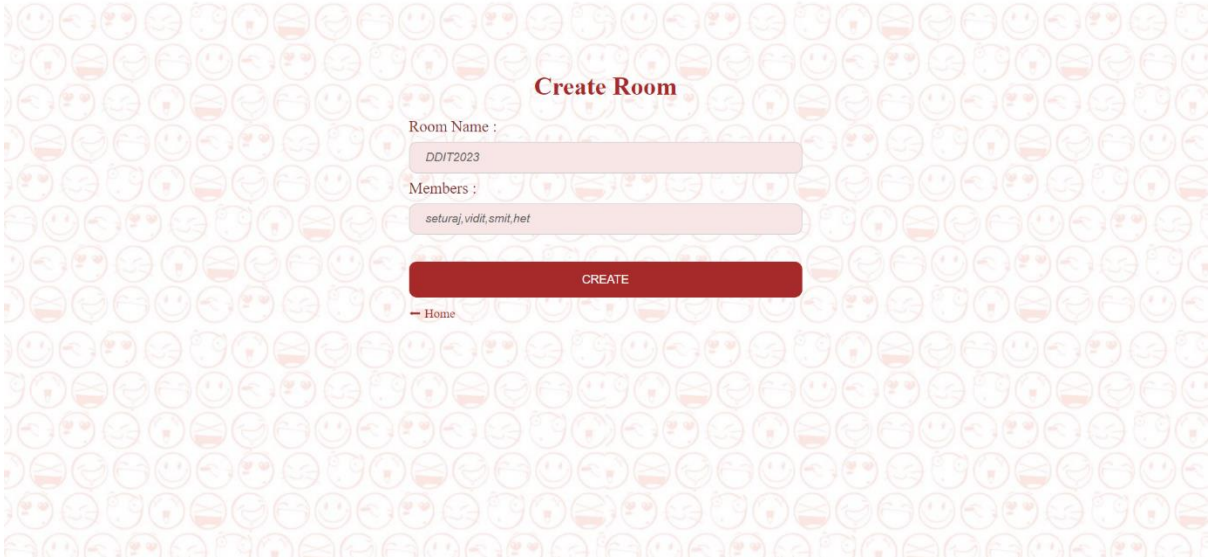
# 7. Screenshots

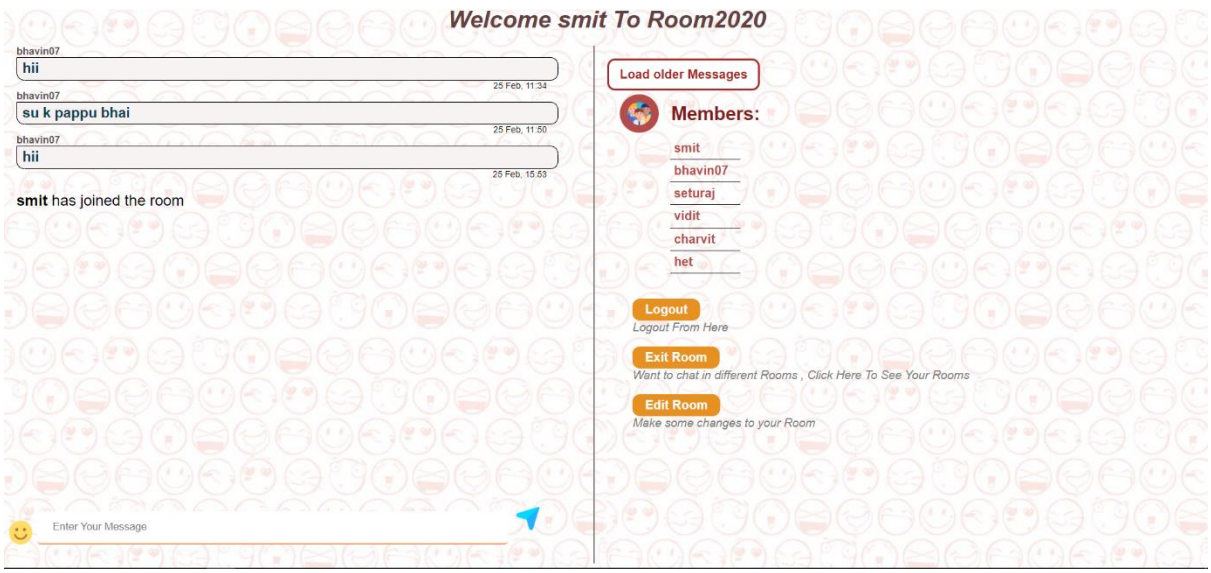


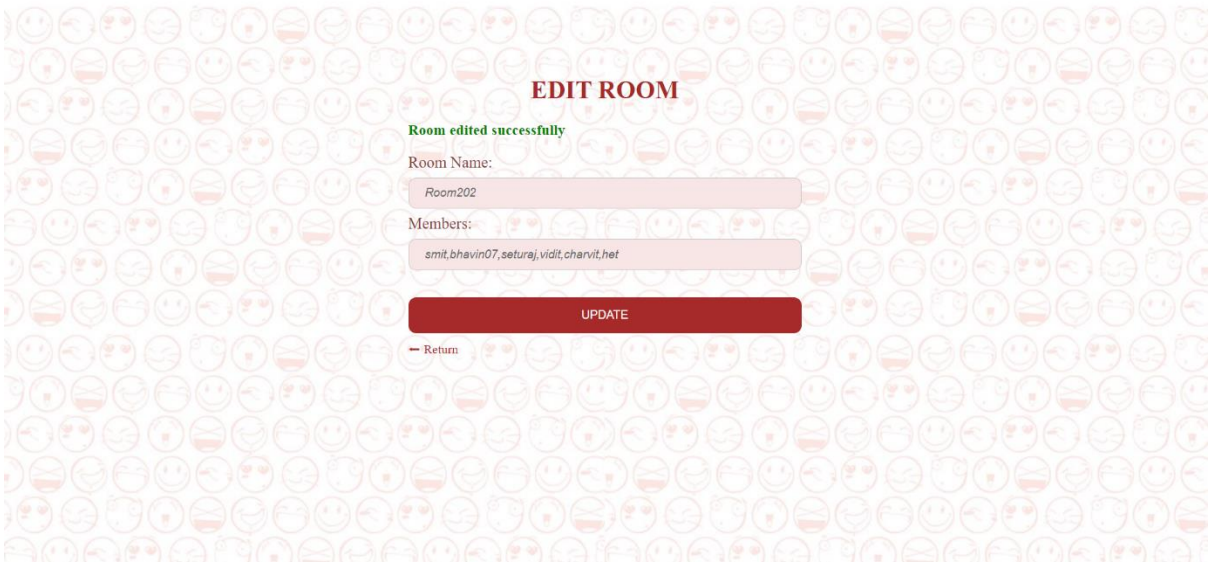



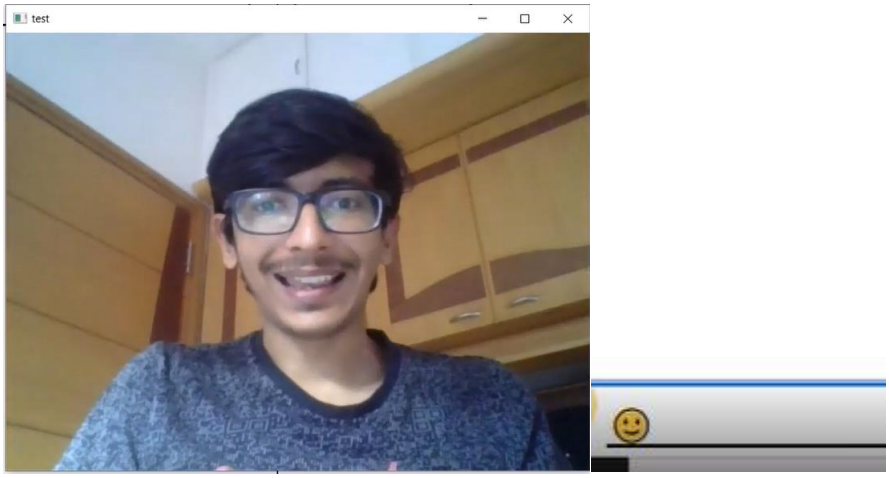



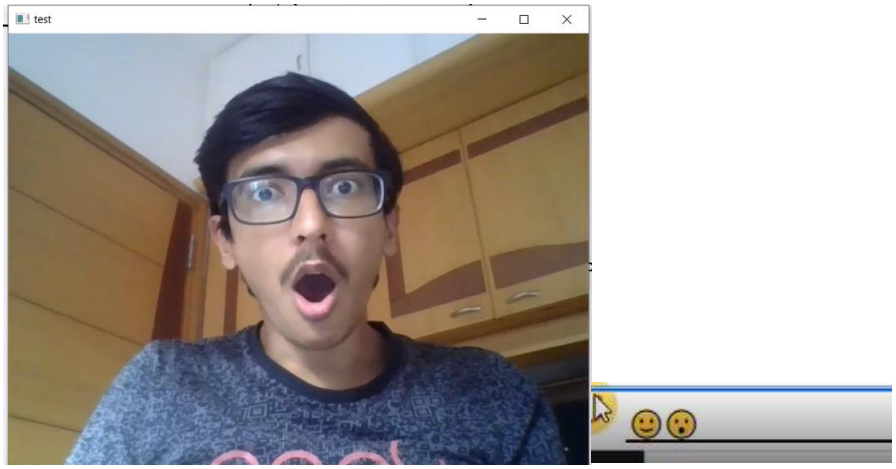

Welcome To our Emojify
Whats Up, bhavin07
Enjoy chatting with your friends
Contact Us
Vidit Shah :
Bhavin Chavda :
My rooms
Room301
Room203
Room303
Room501
Room2020
Dakor
Create-New-room
Create Your Own New Room
Logout
Logout From Here

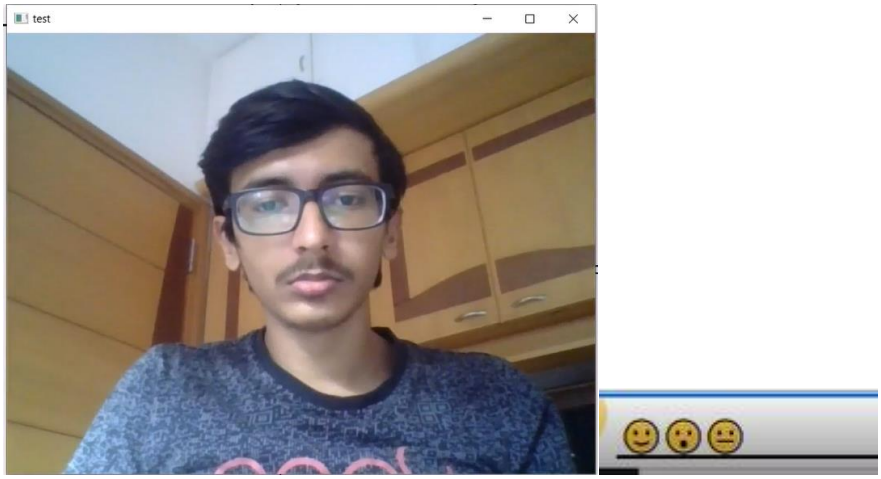

Create Room
Room Name :
DDIT2023
Members :
seturaj,vidit,smit,het
CREATE
Home

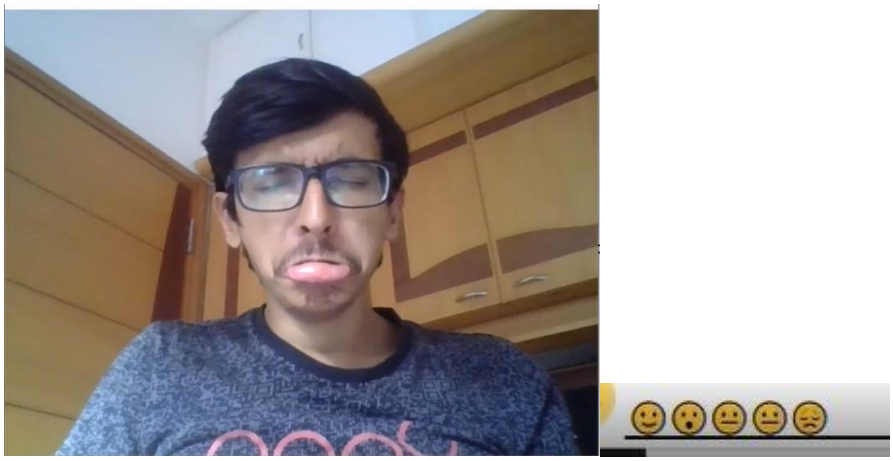

Welcome smit To Room2020
bhavin07
hii
25 Feb, 11:34
bhavin07
su k pappu bhai
25 Feb, 11:50
bhavin07
hii
25 Feb, 15:53
smit has joined the room
Enter Your Message
Load older Messages
Members:
smit
bhavin07
seturaj
vidit
charvit
het
Logout
Logout From Here
Exit Room
Want to chat in different Rooms , Click Here To See Your Rooms
Edit Room
Make some changes to your Room


EDIT ROOM
Room edited successfully
Room Name:
Room202
Members:
smit,bhavin07,seturaj,vidit,charvit,het
UPDATE
Return


test


test



You can watch our emoji testing video-

https://drive.google.com/drive/folders/1H_m0Qz08Dhs6yNzuRKTH3ikbcWJU51Ky?usp=sharing

# 8. Conclusion

The functionalities are implemented in the system after understanding all the system modules according to the requirements. We have built our own model by taking some data from the internet and other by adding our own facial expression to the model. Functionalities that are successfully implemented in the system are:

- Login/Sign up
- User validation
- Create Room
- Join Room
- Send/Receive Message
- Load older Messages
- Recognize facial expression and send emoji
- Edit Room

After the implementation and coding of system, comprehensive testing was performed on the system to determine the errors and possible flaws in the system.

# 9. Limitation and Future Enhancements

The Disgust expression has the minimal number of images – 600 so there is a high possibility that it has the least mapped expression. We have added our own face expression to the model to make it more accurate. Currently, the project runs completely fine if all inputs / selections are given with proper criteria.

In future, we will try to add more emotion to the model and make it more accurate. Add spam detection to the sent message so that the receiver can get notified that this contact is a spam.

# 10. Reference / Bibliography

Following links and websites were referred during the development of this project:


- Training Dataset, https://www.kaggle.com/datasets/msambare/fer2013. Accessed on January 2022.
- Creating Chat Application, https://www.youtube.com/watch?v=uJC8A_7VZOA&list=PLyb_C2HpOQSBUEDI7tx_W4hAz699B6D7p&index=1. Accessed on February 2022.
- For Client server, https://flask-socketio.readthedocs.io/en/latest/. Accessed on February 2022.
- To model group layers into object, https://www.tensorflow.org/api_docs/python/tf/keras/Model. Accessed on January 2022.